Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

DEVELOPMENT OF A TOOLBOX TO HANDLE ONLINE EEG SIGNALS ACROSS THE OPENBCI PLATFORM

Autor: Guillermo Sarasa Durán
Tutor: Francisco de Borja Rodríguez Ortiz

Julio 2015

DEVELOPMENT OF A TOOLBOX TO HANDLE ONLINE EEG SIGNALS ACROSS THE OPENBCI PLATFORM

Autor: Guillermo Sarasa Durán
Tutor: Francisco de Borja Rodríguez Ortiz

Dpto. de Ingeniería Informática
Escuela Politénica Superior
Universidad Autónoma de Madrid
Julio 2015

Resumen

Resumen

Un sistema interfaz cerebro máquina, comunmente conocido como BCI (Brain Computer Interface) es un sistema software y hardware que procesa señales biométricas, en concreto, señales obtenidas de las neuronas y sinapsis del cerebro. Estos sistemas están enfocados principalmente a servir de interfaz de acceso a personas discapacitadas. No obstante, también han sido usados con fines de investigación e incluso de ocio.

Todos los sistemas BCI, de nuevo el conjunto hardware y software, son diseñados para cubrir una necesidad específica dado que las distintas señales de control usadas como "patrón.ª detectar, necesitan de la búsqueda de "trucos"para darles una utilidad real. Como por ejemplo, la detección de patrones familiares para el deletreo de palabras por medio de caracteres parpadeantes. No obstante, las señales de control no son la única variante del sistema, dado que al tratarse de señales biométricas, están plagadas de ruido y poseen un factor variante que puede alterar la producción de las señales de control, y del mismo modo, el éxito del sistema. Esto se debe a que un sistema biológico esta continuamente estimulado y es imposible limitar su interacción con el medio, o incluso consigo mismo.

En este Trabajo de Fin de Grado, como principal resultado se ha desarrollado una aplicación como esqueleto de un sistema BCI sofware propio para el estudio y análisis futuro de la señal. Esta aplicación, proporcionará las bases de una comunicación online entre distintas BCIs hardware, aplicaciones de muestra de atributos y módulos de tratamiento de señal online. Para ello se estudiará, en primer lugar, el estado del arte de las BCIs así como los distintos conocimientos que sean necesarios dada la característica multidisciplinar de esta rama. En segundo lugar, se desarrollará un software suficientemente potente para poder ser reutilizado en un futuro, con resistencia ante problemas de comunicación con el hardware, y suficientemente modularidad para poder sustituir componentes por otros más complejos en trabajos futuros sobre el software. Las funcionalidades básicas de esta aplicación serán un visor gráfico de las distintas señales online y un sistema de predicción simple de la señal de control P300. Para terminar, se valorarán la aplicación resultante, asi como los distintas ramas desarrolladas durante este TFG

Palabras Clave

interfaz cerebro-máquina; progrmación modular con callbacks y controladores; electroencefalografía (EEG); potencial evokado; P300; OpenBCI, BCI On-line

Abstract

A brain-computer interface (BCI) is a software and hardware system, which process biometric signals, in particular, waves from the neuron and brain synapses. These systems aims to help people with dissabilities. However, the BCIs have been used for research and enjoyment features.

All the BCI systems, hardware and software, are designed to suit a specific need due to the different control used like a detect "pattern", needs tricks to be useful. For example, the detection of familiar patterns for a word speller by flashing characters. Nevertheless, the success rate of the system is affected by the noise-signal ratio. This is due to the fact that biological systems are continuously stimulated and it's impossible to restrict it interaction with the environment or even with itself.

In this End of Degree Project, a skeleton application was developed like a own software BCI system, for a future analysis of the signal. This application, will provide bases of a online communication between different hardware BCIs, attribute application and online signal treatment modules. First the state of the art will be studied, and the different knowledge needed, due to the multidisciplinary topic. Second, an strong enough system will be developed in order to be reused in the future. It will endure communication issues with the hardware and enough modularity to change modules for other more complex in the future. The basic features of this application will be: A graphical streaming of the different signals and a simple prediction system of the P300 control signal. Finally the results of the application will be described with the different branches developed in this end degree project.

Key words

brain-computer interface; modular programing with callbacks and drivers; OpenBCI; electroencephalography (EEG); evoked related potential (ERP); P300; BCI On-Line;

Agradecimientos

A mi tutor Francisco, por su continua ayuda en el desarrollo del proyecto, y la gran paciencia que ha estado presente en nuestras conversaciones.

A todos aquellos profesores que me han transmitido el gusto por conocer sin distinciones, y también a los que han hecho por dejar un sistema docente mejor, para todos los que intentamos disfrutarlo.

A mis amigos por haberme ayudado tanto, no solo durante el desarrollo de este trabajo fin de grado, sino también durante toda la carrera. Y en especial a Jaime, Pablo, Sergio, Hao y Javi, por su apoyo, tiempo y paciencia.

Y para terminar a toda mi familia, a los que ya no están y a los que quedan por llegar.

Contents

Chapter 1

Introduction

Nowadays, technology is an important issue in the development of any personal or professional task. In fact, its use is an imperative requirement for an appropriate social interaction, and even a powerful tool for any daily task.

This advance is possible, in a ways, due to the the creation and continuous improvement of the User interfaces (UI) . The UI brings technology for each user, and therefore make the creation of a new brand of services viable in the actual world. But a good UI should be usable and accessible. A BCI (Brain Computer Interface) is a unusual UI, designed for people with disabilities. In this case, the system should be open (hardware and software), and not very expensive. The acquisition price is important because this systems should be accessible for any person. Also, the Opensource feature is critical, due to the future improvement of the platform. Indeed, this features are not enough to make a good UI. The configuration for each user ensures a better user experience. This is achieved by software that adjusts itself to the user preferences.

Thus, the access to a system, determines in some ways its usefulness. But in some cases, the access has an additional problem, which resides in the person. This is the case of the people with disabilities. Hence, the Brain Computer Interfaces were developed to approach technology to anyone. But the development of this technology bring new problems. New technology often means high costs, and consequently fewer people being able to have access to it. On the other hand, a better BCI means a bigger hazard for the subject, due to the invasive methods to extract signals from the human body. Additionally, a BCI provides a specific communication, and in some cases needs a training both to learn to use it and for every time the user wants to use it.

This end degree project aims to be an introduction to the BCI world in order to be the beginning of future projects. The target will be to develop a software which could communicate between different BCI hardware platforms and user applications. This software will provide a base with simple tools (e.g. a GUI for data stream) and a modular construction to interact with any BCI hardware (e.g. OpenBCI). Moreover, a simple events classification system will be developed to study the main features of the system.

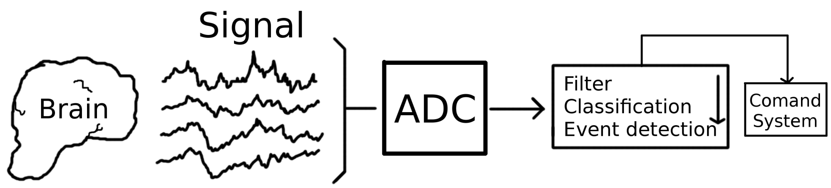

Figure 1.1: Simple scheme of a BCI system. The brain signal, the recorder; the Analog Digital Converter, the biometric data handler; the filter and classification system; and the Command system

1.1 Objectives

The objective of this end degree project, is to get to know the BCI world and the different disciplines needed for this area of knowledge. Additionally, the application result of this project will be used in future ones, in particular for a *close loop* system [16] in the context of BCI systems.

In order to achieve this, the following sub-objectives were established:

- Develop an own interface and signal capture system, to plot the EEG signal. This will be useful to study the features of the signal with online platforms.
- Design a modular system to incorporate the future BCI SDKs of different systems for EEG data acquisition.
- Implement a multi-thread system strong enough to resist CPU latency.
- Build an introductory P300 detection system. The purpose of this system is to face a P300 control signal and it associated problems.
- Develop an introductory online signal analysis.

Chapter 2

State of the art

2.1 Brain Computer Interface (BCI)

2.1.1 Description

A BCI is a system that takes a biometric signal from the brain and creates an interaction with a Computer ([11] and [2]). It includes hardware and software platforms, each one of them divided into different subsystems (see figure 1.1):

The recorder
It registers the signal from the human body (e.g. electrodes).

The biometric data handler (ADC)
It manages the measured data (important to establish a safely separation between a power source and the subject).

Filters and classification system
This system processes the signal data. First, it applies some filters to it to eliminate noise, and to clarify the signal. Once the signal is filtered, the system tries to classify it. Usually there are two classes of events: Desired and undesired. But in some cases it could be useful to have more than two classes (e.g. sensory sensorimotor rhythms: Right hand movement, left hand movement, no movement).

Command system
The command system translates the instructions from the BCI classification system to real actions (e.g. move a wheelchair).

Thus, the implementation of each subsystem determines the accuracy of the BCI.

2.1.2 Types of BCI signal recorder

The integration of modern technology in the human body has its limits. More resolution means more health hazards, hence there is a risk/resolution relation in any BCI system. The BCIs are separated into two initial types according to this relation:

2.1.2.1 Invasive

An invasive BCI represents a system that has sensors inside the human body. This systems are more dangerous but have more precision than the Non invasive. The extra temporal and spatial resolution is around 100 times better, and, in average, the Invasive systems are more useful and faster. An example of an Invasive method is the ECoG [11], which uses internal sensors to get a better signal-noise rate.

2.1.2.2 Non invasive

The Non invasive system represents an useful solution with reasonable results and zero risks. Although the low resolution implies less information per time, this is enough for some problems (e.g. Speller program, oddball paradigm [5]). On the other hand this kind of BCIs need less training and configuration to extract the desired control signal.

In conclusion the Non invasive represent a reasonable solution with a near *plug-and-play* functionality that gives the user more freedom of use. This system is used in this project because it provides a usable and accessible hardware BCI method.

2.1.3 Types of signals

There are different BCI signals, related with each type of signal recorder. The following are a little example of the different types of signals that can be studied for the development of a BCI.

2.1.3.1 Electroencephalography (EEG)

The synaptic function of the brain causes traces of electrical currents that could be recorded with the use of a non-invasive BCI. Nevertheless, the scalp, hair, or even the own activity of the brain of the subject, could produce noise in the record. This represents a problem in the classification of the signal. To avoid this problem, the BCIs systems should implement previous signal filters.

The signal recorded is measured as an electric potential difference of the continuous signal and the electrode reference (ground reference to be calculated). Thus, the sensor configuration is simple and doesn't require special care, while the position of the sensor does not change. In order to guarantee their position, the electrodes are placed in helmet structures. The position map of the electrodes is defined by the 10-20 system [15].

The electrodes for the sensors can be of different types, which determines a little the quality of the signal as is described in 2.1.4.

2.1.3.2 Magnetoencephalography (MEG)

MEG and EEG have the same behavior and record the same synapse traces, with a difference in the measure method. Instead of recording electric potential difference, it records the differences of the magnetic field [11].

The change in the measuring method reduces the noise and produces a more clear signal, but it needs a less accessible method of BCI recorder. A MEG BCI needs superconducting quantum interference devices (SQUIDs). But the equipment needed for a

SQUID has to be cooled at 0º Kelvin (to maintain the superconducting property), and need special magnetic isolation.

For these reasons, a MEG BCI is not a good option for a people orientated BCI.

2.1.3.3 Electrocorticography (ECoG)

In the way of the two last signals, the ECoG measure the synaptic traces with internal electrodes placed in the brain cortex. This electrodes provide a higher resolution but with the health hazard of the internal electrodes implant operation [11].

Although the reject problems are solved, the maintain of internal electrodes and their connections with the BCI, make ECoG a no viable option for a people orientated BCI.

Others signals There are other types of signals (e.g. intracortical neuron recording, fMRI (functional magnetic resonance imaging), NIRS (near-infrared spectroscopy)) more difficult to measure, but, in some cases, with better results [11].

2.1.3.4 Conclusions

EEG signals have poor noise-signal ratio, but without risks and easy to extract, configure and maintain. Summing up, EEG has good features to be classified as a reasonable accessible and usable option.

2.1.4 Types of sensors

There are several EEG recorder methods, with different signal resolutions and placing methods. The clearest signals are usually found in Gel electrodes. This electrodes use a conductive paste or a conductive solution to improve the signal. The gel and wet electrodes can be passive or active. The difference between them is the use of an amplification system. However this systems are not welcome by some users. The alternative are the dry electrodes, which use no solution, but have more noise. For this reason, the dry electrodes are more usable. Nevertheless, the signal has enough resolution to detect some control signals [10]. Additionally, any electrode can use an additional power source to amplify the signal (active electrodes).

2.1.5 Control signal types

The chosen control signal determines the possible uses of a BCI. For that reason the use of a BCI is not as useful as it could be. Each control signal uses a different pattern to classify the signal, which means that each pattern needs an application which works according to it (e.g. detect the moment a person blinks).

The easiest control signal is the recognition of alpha waves, which appear when a person closes his eyes during a very short period of time. It causes a very soft line in the back sensors, which is very easy to recognize at plain sight. However, all this control signals increment their difficulty with the BCI resolution, person and even the state of the state of the person. The biological systems are sets of complex tasks which, in some cases, are coordinated by the brain. Thus, classifying a signal captured from the *traces* of the brain sourced from a singular task is not a simple task. Additionally, each control signal has different sensors positions that provide different levels of quality. For some configurations, the control signal could be impossible to measure.

The synapses produce different EEG signals depending on some external and internal factors (temperature, fatigue, hunger…). Hence, the action-reaction relation between a signal pattern and an event should be big enough to be "easily" classified in noise free conditions. The following are some examples of different control signals.

2.1.5.1 Steady-State Visual Evoked Potentials (SSVEP)

SSVEP is a type of Visual Evoked Potential (VEP) [11]. VEPs control signals try to classify modulations in the brain activity after a particular stimulus. The Steady-State VEPs (SSVEPs) are based in a set of sources with a particular behavior. Each source behavior is chosen by its score in a previous training (in some cases). This control signal is relatively easy to detect, but has it limitations. More targets means harder classification systems and hence less bit/sec ([3] and [4])

For this project the SSVEPs signal will not be studied because it was analyzed in [16]. In that project a *closed loop* system was developed to adjust the signal pattern recognition to each subject. This idea will be studied for a P300 compatibility in the future work of this project, as is described in section 4.4.

2.1.5.2 P300

The P300 control signal searches for a particular increment of the EEG signal 300ms after an unusual event appears (e.g. a series of pictures, in which you recognize only one of them). But the time elapsed between pictures or recognizable events introduces a problem. A long period of time means a slow communication between the software and the person. On the other hand, a small period of time could prevent the P300 event from happening. If the subject doesn't realize after seeing the recognizable picture, the P300 would not appears in the recorded signal.

The electrodes used in P300, according to the 10-20 definition [15], are: Fz, Cz, Pz, P3, PO7, PO8 and Oz. The reason to restrict the electrodes used, is that the EPR apparition doesn't manifest in all the sensor signals. However, this electrode positions are only a guideline. Each subject could manifest different behavior [6].

The standard definition of a P300 signal is represented in figure 2.1, where a voltage increment is shown after 300ms. But in some cases, the P300 is not as simple as it is defined. With the figure 3.2 we can see the difference between a theoretical signal and a real signal taken from the competition data explained in the section 3.1 used in this project. The figure 3.2 represents an average of a set of signals where a P300 appears.

This control signal was defined in the oddball paradigm experiment.

Oddball paradigm: The oddball paradigm defines an experiment used to classify a set of visual/audio stimuli to detect the desire one [5]. The most common example is a speller matrix (figure 2.2), which highlights its rows and columns individually in order to find the desire character. This search is called *character epoch*. In the oddball paradigm so many character epochs are defined to spell words. The control signal to be classified for this problem is P300. In each character epoch, one row/column stands out at a time. Once

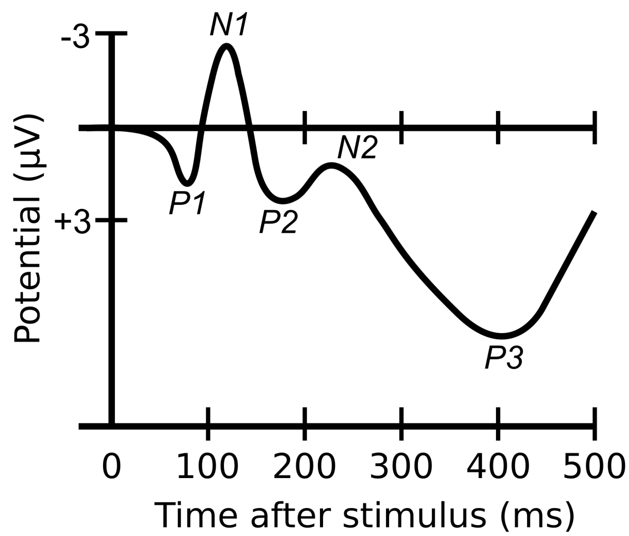

Figure 2.1: EEG signal recorded from a BCI 500ms after stimuli (Adapted from `https://en.wikipedia.org/wiki/Event-related_potential`

each row and column are shown a number of times, the BCI system should return the target character.

2.1.5.3 Other signals

There are different control signals like sensorimotor rhythms, slow cortical potentials or other VEPs types. For this project, the control signal chosen was P300. However, the classification system aims only to try to find features for a future work with the software.

2.1.6 The objective

A BCI aims to solve access problems for people with disabilities, but it could be used with others purposes. Thus a BCI is above all an accessibility tool with a very precise task.

2.2 BCI platforms

A BCI platform could be implemented in software, as a signal process; or as a hardware tool to record data.

2.2.1 Software platforms

There are multiples solutions for solving the software classification needs of a BCI system as its described in [13].

BCI2000: An Open Source platform with Matlab interaction, which is compatible with C++ and python interfaces. It can provide a real time signal process. It only works in

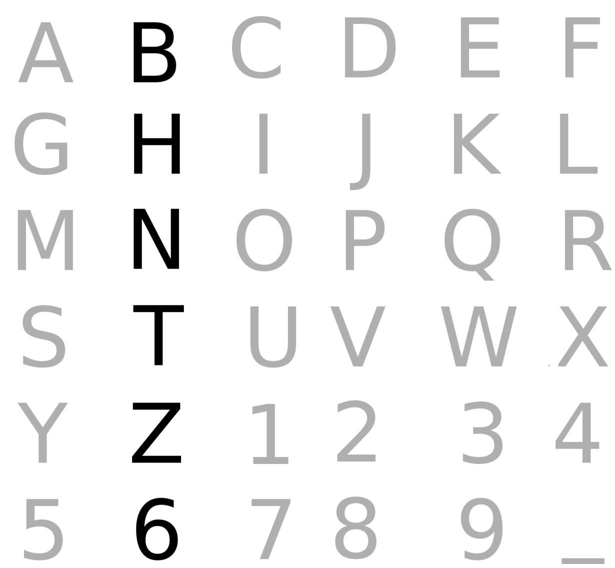

Figure 2.2: Speller matrix commonly used in the oddball paradigm problems

windows systems and with standalone BCI appliance, and would need a powerful machine to run it. For some BCI communications, Borlands C++ knowledge is necessary.

OpenVibe: OpenVibe is a open source platform for BCI control and signal processing. It has an easy configuration for users without coding knowledge, with a extensive GUI, which provides a modular structure expansible with new features.

BCILAB: Oriented for Matlab use, it has a GUI and a scripting UI, with a fast configuration method, that minimizes the time for a quick test. It can manage 32 channels and plot 3D models (e.g. the brain activity in a 3D map).

BCI++: This is another Open Source solution programmed in C++, with a powerful GUI that can interact with the XNA module of C♯. This application is divided in two parts: The Hardware Interface Module, which manages the acquisition, storage, and real time processes; and AEnima, the GUI module.

Other solutions: xBCI, BF++ ,Pyff...

2.2.1.1 Conclusions

This end degree project will develop a new software to understand the main features of the EEG signal, and the problems related to the detection of a P300 signal. Additionally, the development of a main tool to both manage and classify EEG data will be useful for the development of a self-configuring application [16]. It will modify its actuation flow and classification system online. Thus, instead of configuring the system for a unique solution or a greedy solution with bad results in some subjects, the systems will configure it-self.

This concept is called *closed loop*. In order to study this concept, it is necessary to use a new BCI software, designed for this purpose. The application requirements are described in section 4.1.1.

2.2.2 Hardware platforms

There are a lot of BCI hardware platforms, but the access to the best BCIs is reduced due to the expensive cost of this platforms. On the other hand, those with less quality and costs, could be used in the same way, but with better software support.

For this reason, this end degree project has the purpose of developing a solution using one of this accessible pieces of hardware. The following platforms are three examples of the most accessible BCI hardware platforms (according to the accessibility definition in section 1):

2.2.2.1 EMotiv EPOC

EPOC [7] is a 16 channel hardware that could be used for the construction of a BCI. It uses wet electrodes and has a solid structure in order to ensure they stay in an appropriate position. To check that, EPOC has a simple stream data software, which has a electrode position feature. However, the use of that structure doesn't allow for the use of gel or dry electrodes. This platform is not Open Source Hardware (OSHW), which is essential for a BCI according its description in section 1.

2.2.2.2 Olimex EEG-SMT

On the other hand, OpenEEG is a 5 channel hardware solution OSHW (showed in figure 2.3). In this case, the user can choose the electrodes to use, but the hardware limitations only support 5 passive electrodes or 4 active electrodes and 1 passive. It is being developed under the OpenEEG project, which has multiple software and hardware solutions to build a BCI. This represents a cheap solution in continuous development but with big hardware limitations.

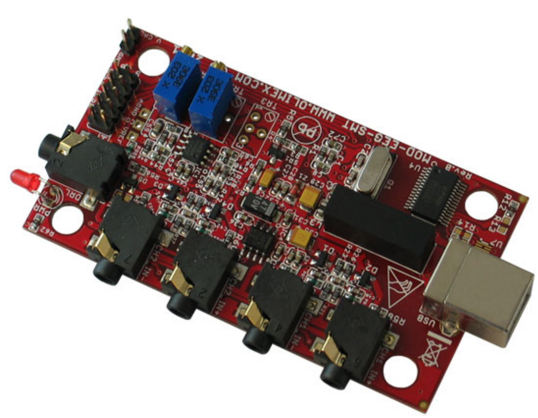

Figure 2.3: EEG-SMT Olimex (Adapted from `www.olimex.com/`)

2.2.2.3 OpenBCI

OpenBCI is a recently born open hardware and software platform. This platform is currently under continuous development. The last version of the hardware is the 3rd (see figure 2.4), available in two versions (8bits ArduinoUNO, 32bits Chipkit). It supports at least 8 channels (16 with an extension module in the 32 bits version). Moreover, the continuous improvement of the SDK by the open source community makes the OpenBCI more compatible with other software platforms (e.g. OpenViBe). This is a cross-platform software with two different versions developed in JAVA and python, with C++ and javascript versions under development. The JAVA software is oriented to a GUI streaming interface in Processing (an advanced GUI orientated language).

The current version of the OpenBCI has a 3D model headware to place the hardware and the electrodes. This open source 3D printed headset has improvemed its structure under the community's support. Additionally, it is oriented to work across external software, which makes it a good open platform choice.

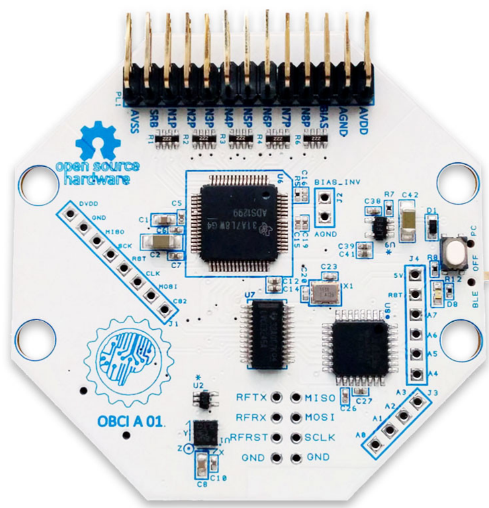

Figure 2.4: OpenBCI 8bits ArduinoUNO version (Adapted from `http://openbci.com/`)

2.2.2.4 Conclusions

The previous hardware platforms represent cheap and functional solutions, but the electrode limitations of the Olimex, and the private SDK of EPOC, make OpenBCI the best hardware solution.

2.2.3 BCI Data

One of the problems of a BCI platform is the selection of the algorithm and learning systems to filter and classify the desired control signal. In some cases, the initial conditions determine the success of the selected filter and classification systems. Hence, different configurations means different results. For example, the oddball paradigm has different matrix representations (see figure 2.5).

Figure 2.5: An example of a different representation of the oddball paradigm

Thereby, to solve the detection problem a previous research is needed. In order to achieve this goal, it is helpful to minimize the number of SPOF (Single Points Of Failure). For that reason, before developing an online BCI system, the offline data are a good start test for the study. One of the most important sources of BCI data is the BCI competition. In this competition, some of the initial variables (number of channels, record protocol. . .) are fixed. So the goal is to find the best solution for a data set, with the minimum amount of training data. In the case of this end degree project, the competition chosen was the III BCI competition, II problem [12].

Chapter 3

Methodology

For the correct development of the project, it is necessary to establish a methodology adapted to the problem. Hence, this methodology should affect the data processing, communication system and code standards.

3.1 Data Source

The online data analysis is a complex task which requires a previous knowledge. To remove the uncertainty around the online process, it is necessary to previously study offline data. The offline solution chosen for this project was the BCI competition [12].

3.1.1 Competition data

The source of the data was from the III BCI competition of 2004 ([12]). The recorded data includes the event's data and the signal recorded from the BCI with the following format:

$$\textbf{Samples} \times \textbf{Character Epoch}$$

A character epoch represents a set of samples with an common target character. The duration of one character epoch according to the competition description is 32.5 seconds, although in the recorded data some samples are not included.

Each character epoch has an *atomic unit* defined in figure 3.1.

The data was recorded flashing each row or column for 100 ms followed by a pause of 75ms. This is repeated 15 times, due to some times the ERP doesn't appear. Thus the desired character was flashed 30 times. There is a pause between character epochs of 2.5 seconds, which is not represented in any data included in the set.

The data was recorded at 240Hz with 2 different subjects, with different sessions for each one. Each session has a training and a test data set, with the following files:

Flashing
: This file shows the samples where a row/column was intensified (1,0).

StimulusCode
: Like the Flashing file, it shows the intensification of a row/column but with the code of the row/column.

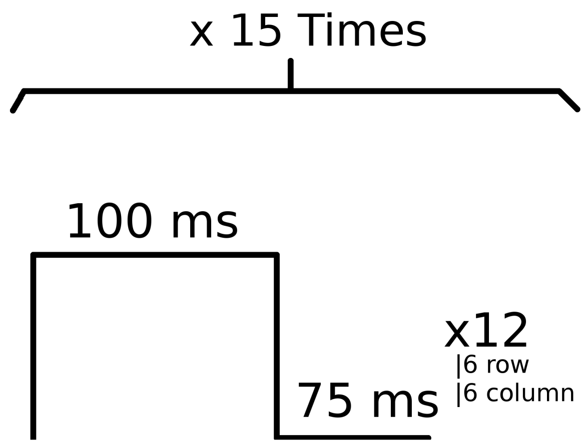

Figure 3.1: Atomic unit of the competition data set

StimulusType
This file shows the sample where the desired row/column was intensified.

TargetChar
The desired character for each character epoch.

The differences between the training and test data sets are the number of character epochs (85 and 100, respectively) and the two last files. This files are only included in the training data set. All this data is stored in different .txt files.

Finally there are two signal files (training, test) with the following format:

$$\textbf{Samples} \times \textbf{Character Epoch} \times \textbf{Channels}$$

This files store the recorded session for each sensor (channel). An example of an averaged session is included in figure 3.3. For each row and column 100 measures are taken at 240Hz. Each field is the averages of the measurements of its row and column. This figure shows the difficulty in detecting the P300 control signal. An event-related potential (ERP) should appear around the 72^{nd} sample (sampled at 240Hz; the sample 72 is equivalent to the 300ms). The target field is the 'I', so each field of the second row and third column, should present a similar behavior. But other fields share this behavior too, hence the visual component of the P300 problem, is not as helpful as it have to be.

3.2 Signal treatment

As the we have just seen, the task of detecting an ERP requires a complex filter and classification system. For this case, the detection system was built upon the data of the competition described in the previous section, so it may not work with other BCI data inputs.

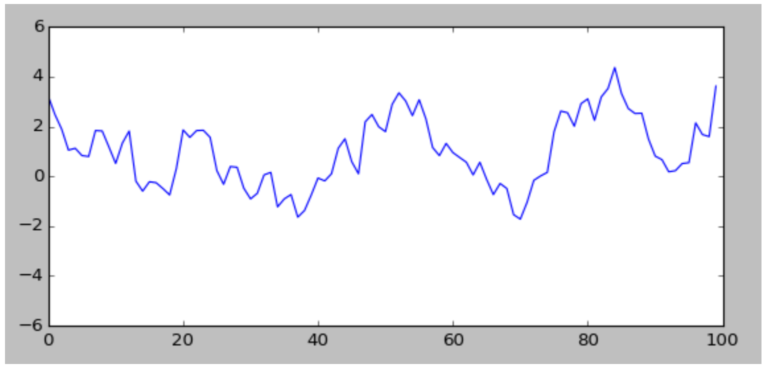

Figure 3.2: P300 epoch sampled at 240Hz (taken from the data set)

3.2.1 Previous analysis

To determine which filter, classification or variables will be optimal, a previous analysis is necessary. This system will provide a tool to analyze the signal visually, search for the pattern of the desired event, and compare it with the others. However, to analyze this problem, an external tool was developed. This tool shows a matrix with the average of the signals for each character epoch (see figure 3.3).

According to the P300 description, the signal has to modify it flow with a soft voltage increment whenever the desired attribute is shown, but in this data case, the increment is really difficult to find due to the fast interval between row/column flashing (100ms). The standard P300 signal behavior and the behavior of the Competition data set suggests the need of a previous filter.

As the figure 3.2 shows, the P300 displays a noisy signal. To face this problem the data were arranged in the form of a matrix, as describe previously (see figure 3.3).

Finally, with the double average classification it is possible to determine visually the desired character about half the times. For this reason, the average matrix system needs an additional classification system.

3.2.2 Filters and features selection

The reason to apply filters [11] and features selection and extraction methods [11], is to delete irrelevant information. The main difference between normal filters and the features handler methods is the complexity of the information selector. These are some examples of filters and features selections used in the development of the detection system:

Bandpass filters: This filter is used to extract the noisy frequencies of the signal. This is a common method used in some BCI competitions to obtain a more clear signal. In this project the signal was bandpassed using the Fourier transform. With the Fourier transform a signal can be represented as a frequency specter. Once applied, the undesired frequency bands are deleted. Finally the inverse Fourier transform operation is used to recover the bandpassed signal.

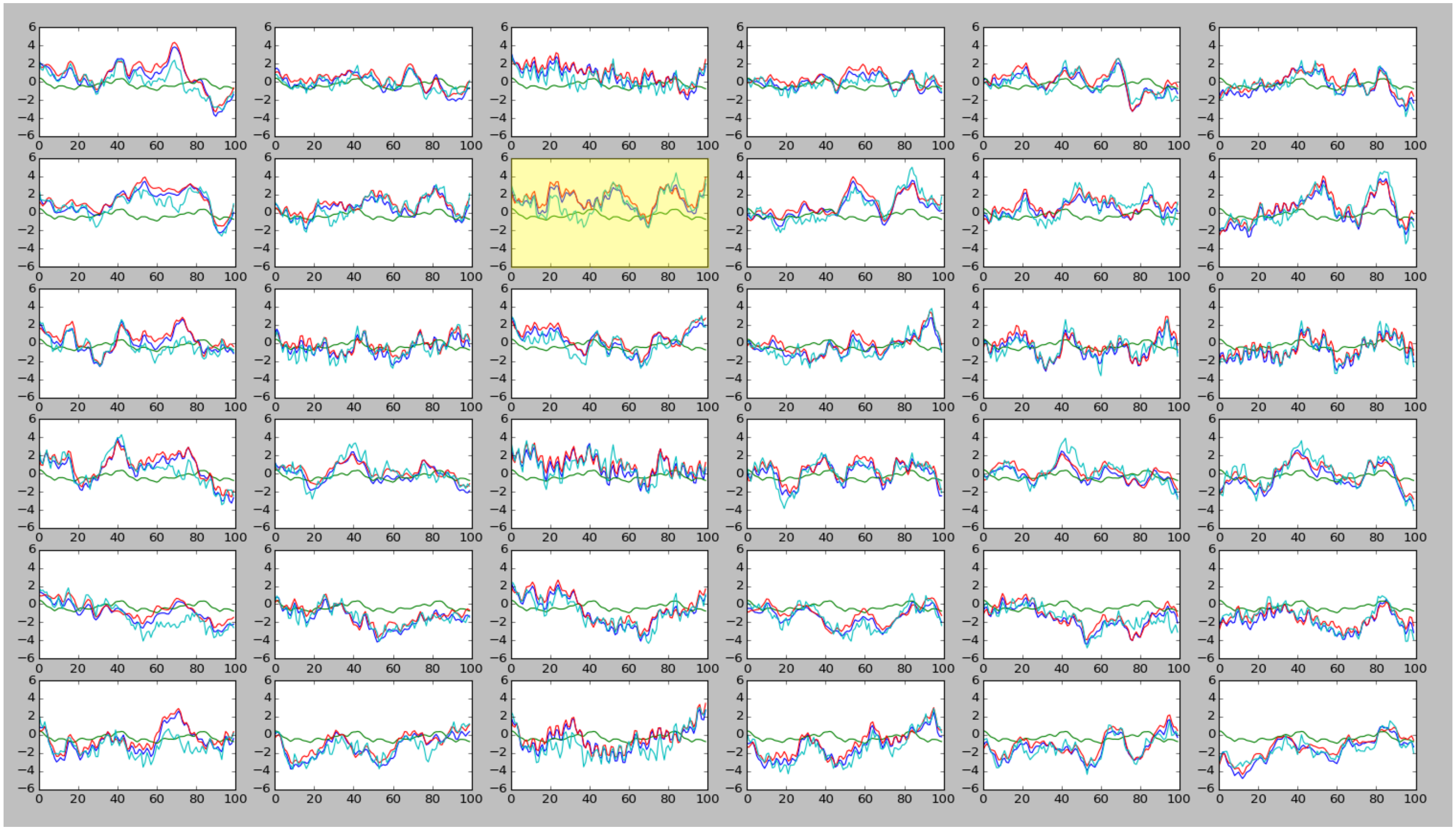

Figure 3.3: Speller matrix representation for the EEG data. The ERP field is the **yellow** one. Each field shows different average of events signals: The average of each recorded signal for this field (row and column) **blue**, the average of each recorded event signal **green**, the difference between green and blue **red** and the median of each set of event signals **cyan**

Principal Components Analysis (PCA): This technique redefines the n-dimensions given in the input data. An easy way to understand this feature selector system is with a 2-dimensional data. For example a EEG with 2 sensors. The PCA will redefine the axes centering both in the averages of the old ones. Their orientation would depend on the dispersion of the data. In figure 3.4 there are different points with no apparent relation for a 3 Dimension data sample. Once the PCA is applied to the data, the output dimension has changed, giving a different values for each one. The reason of use this feature selector was inspired by the [18]

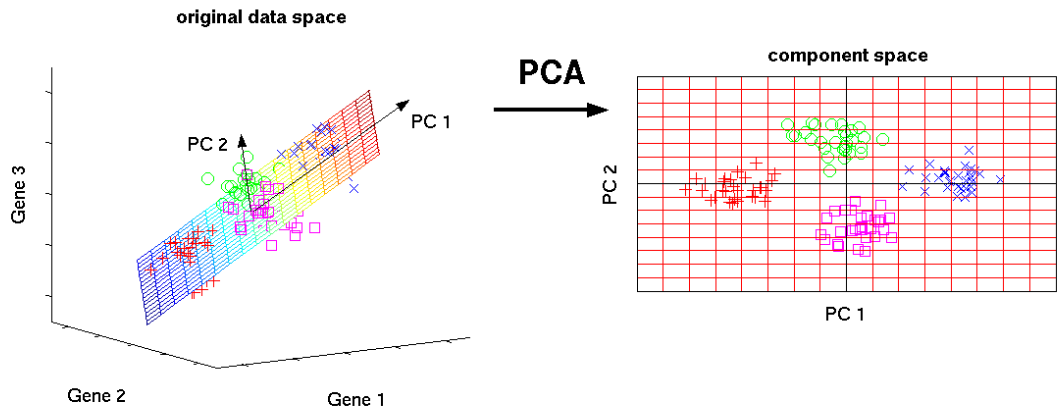

Figure 3.4: Simple scheme of PCA features (Taken from `http://phdthesis-bioinformatics-maxplanckinstitute-molecularplantphys.matthias-scholz.de`)

This technique is used in EEG analysis to try to isolate a pattern in one dimension (e.g. sensorimotor rhythm).

Common Spatial Pattern (CSP): This is a more precise method to classify data in particular circumstances. This method takes two covariance matrices, each assigned to a class. Thus, CSP builds the normalized spatial covariance matrix. This matrix will be used to multiply each data signal. The result should have a maximum variance for one class and minimum for the other. However, to obtain a good result this method needs many entries for each covariance matrix. The reason of use this feature selector was inspired by the [19]

For the data used in this project, CSP gave very good results in train test, but the overfitting makes the test error go over 70%. This results were reasonable, due to P300 (and specially in the competition data) having a sensor limitation, as is described in section 2.1.5.

3.2.2.1 Support Vector Machine

A SVM is a machine learning system that creates a hyperplane or a set of hyperplanes to separate the optimum samples in classes. Thereby, this system needs a training set to build this barrier. With this data, the SVM finds the hyperplane that maximizes its distance with the nearest points in order to achieve better results (Figure 3.5). However, some separation configurations create an overfitting in the SVM. To avoid this, the SVM has a parameter C to create soft margins. This usually increments the training error, but decrements the test error.

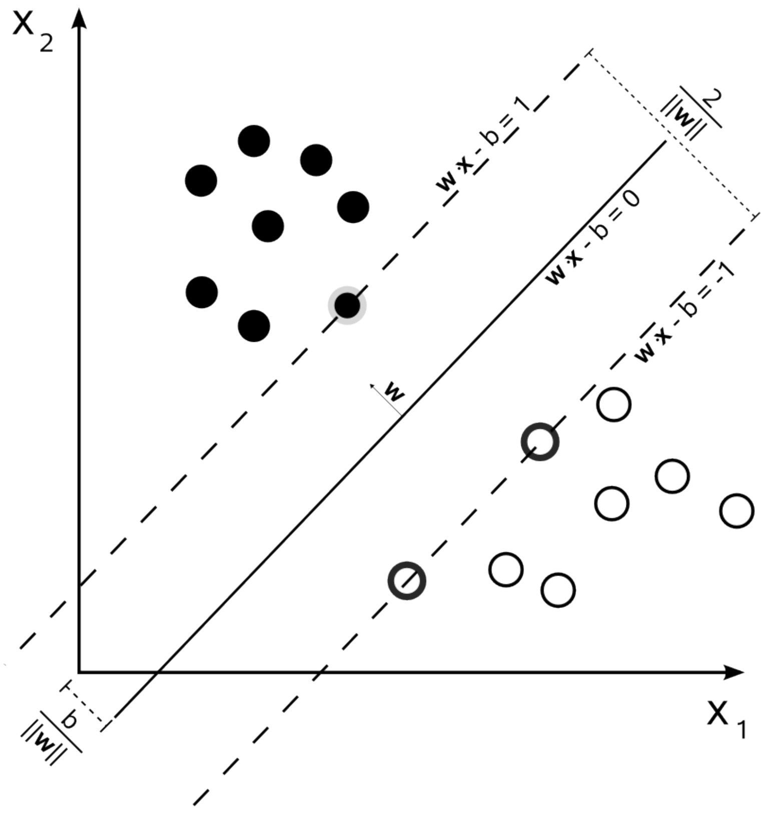

Figure 3.5: Simple scheme of the SVM (taken from `https://en.wikipedia.org/wiki/Support_vector_machine`)

The development of the machine learning system has been inspired by the winners of the BCI competition ([12]) and the common use of SVM in BCI problems ([17]). The solution chosen was the SVM system of Alain Rakotomamonjy [14]. The winner system defines a Chebyshev Type I Bandpass Filtering with a 8-order filter which cut-off frequencies are 0.1-10Hz, decimated, and a SVM with different parameters to obtain results. After the classification, they define a 36-class problem, in order to detect the P300 events. The 36-class problem was built using and ensemble of classifiers system, with different data for each classifier. Each training data set was built assuming that there is less signal-to-noise ratio among the characters of a word. Then, they make different training sets for each 5 character set. This system is a full detection one, with enough complexity for another end degree project. Thus, a simple prediction system to test this feature was developed. This uses an bandpassed signal and a Gaussian SVM.

Furthermore, other SVM systems were developed to find the best system (for example B). The best system is defined as the system than minimizes the errors with the minimum amount of train data. In this case the train data are essential, because they will be taken in real time with each subject with the use of a online BCI.

3.2.2.2 Variables simulation

The configuration of the different learning systems was developed with a simulation system. This system was developed at the same time as the main project.

At the beginning of the project, the simulation searched for the best values of window width, train percentage (quantity of data dedicated to training), C and γ (for the Gaussian SVM configuration). But the time required for this simulation was too long, so some variables were fixed.

These simulation systems implement a bi-dimensional window search for the values of C and γ. It builds a grid and uses it to search for the best values in the line cross points. Once the optimal points are found, the system executes the simulation again around the new point. The train percentage and the signal window width are searched individually. The search was implemented recursively. Thus, the test of a value of one level needs the execution of all the recursion of the next variable. The depth of the recursion search is defined for each variable search function, and it can be changed. After testing the value of one variable, the function continues testing the values for all the interval defined (in the bi-dimensional search, the points of the grid). After the end of the search, the function calls itself with a new interval which size is twice its width (see figure 3.6). Once the final depth is reached, the function checks the returned value in order to find better results. This process needs special care, because of the time needed for its execution.

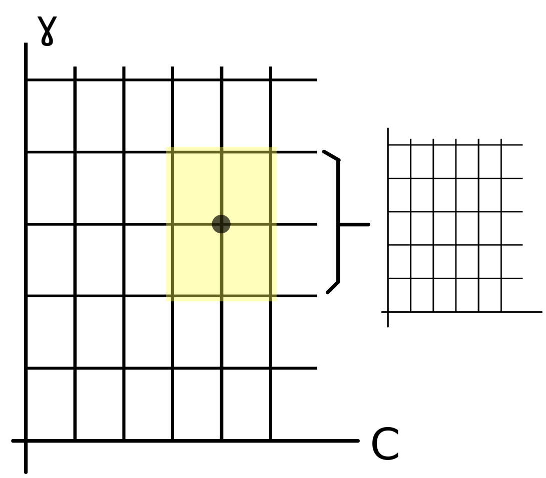

Figure 3.6: Simple scheme of the simulation with a bi-dimensional search system for C and γ

This simulation system was used for other variable searches. For example, the limits of the Peek Algorithm (described in section 4.1.4), which uses an average system to define the width of the window defined by this limits, were iterated to try to find a better configuration. Additionally, to minimize the simulation time, some parameters of the configuration were fixed.

3.3 Callback modular method

One of the objectives of this project is to achieve a software which can work with any BCI hardware platform with little code implementations to communicate with it. To make the software compatible with any BCI hardware a modular design (abstraction of methods) and an asynchronous component are needed. For that reason, the callback methodology was chosen to implement the communication between modules. The main features of the callback functionality are:

Modular implementation
: The callback method provides a modular communication system which creates a transparent communication. This is possible thanks to the fact that the launched threads receive a callback as an argument parameter. The communication can be established with no issues.

Abstraction calling method
: The previously described method also gives abstract functionality to the program process. This is important to integrate new BCI SDKs.

Asynchronus communication
: The callback asynchronous communication is important for the online communication. The use of synchronous communication implies more control over the communication between the BCI and the application. This makes the communication slower and could create problems in the future with different BCIs.

Execution in different threads
: This is essential to optimize the online process (execute multiple threads "at the same time").

Busy fault tolerance
: Sometimes a busy CPU can create latency in the response of an application. This software must resist this kind of problems because it needs a long period of time to answer (prediction system). The callback functionality creates the possibility of reading and processing data separately. Thus, the BCI SDK can read at one frequency, and the process thread only processes under the CPU capabilities without any error. The figure 4.5 shows this problem. The GUI showed in the picture uses *timers* to check the new data (where the min value is established by the plot latency).

The callback flow is shown in figures 4.3 and 4.4.

3.4 Driver development method

One of the requirements described in the section 1.1 is the capability of using different modules for one purpose without need of extra code in the rest of the application. For that reason, the connection of the different modules were oriented to a driver system connection.

This methodology defines a few functions for the communication of each module. This functions implement the communications driver, and every implementation should share the communication data structure (I/O data). This is important because future work will use different BCI hardware and software solutions. The driver specification is applied to the BCI, the signal treatment driver and the attribute software (application)

3.5 Working method

In this project, the most part of the coding time was used to test different systems and find out which one of them gives the best score. For this reason, the development of the project was shaped according to this problem.

3.5.1 Development methodology

The workflow was divided into two phases: The first step was research, to study the features of the signal and learning system. Additionally, in this phase, some simulation was executed in order to find the best configuration of the system without the problems of a sequential workflow. In the second step, all of the code needed for the final software was encapsulated and integrated.

Scripting research method: Python was the programming language used. It has many useful interfaces to work with. The most used during this project was *iPython*, run with Python2.7. iPython saves all the context of the executed code. Thus, the code could be modified near the execution time to study the signal correctly. This working method was useful to minimize the SPOF (Single Points Of Failure) and the issues of the common "trial and error" programming method.

Encapsulation and integration: Once the earlier scripting functionality was finished, it was encapsulated in the corresponding module. This applies to the more complex development tasks. The ones which are related with the internal function of the application were implemented in the second phase.

Chapter 4

Results

To fulfill the purposed objectives in the previous chapters, and taking into consideration the specifics of the problem, the following application was developed. It is a simple introduction to a future work in the subject, being a basic application that provides its structure.

The main feature of this software is the capability of an online code workflow. This is indispensable for the future development of *close loop* algorithms in the system. This is due to the fact that these algorithms aren't compatible with other software platforms (see section 2.2.1). The current version of the project implements a streaming GUI signal and a simple train and prediction system (see section 4.1.2.1)

The connection with the OpenBCI hardware is established thanks to its SDK. This software provides two different ways to get data from the hardware described in the section 4.1.3.1. The first one shares the callback structure with the project specification.

4.1 The Application

The main design of the application was oriented to a modular driver based system. Thus, the BCI modules and communications were built according to a driver definition, which provides new features transparent to the rest of the software, as its described in the section 3.4.

The modules that implement the driver functionality are the signal treatment driver, the application, and the BCI. The first uses main functions to define the learn and predict methods, but it can be modified to use different classification or filter systems. In the case of the BCI, the driver feature defines streaming methods, and each BCI can implement it according to its needs. Finally, the application module used is defined at the beginning of the execution. It is independent of the rest of the modules, which is important due to the possible interaction with different configurations. This interaction is necessary in order to improve the apparition of P300 events. These features are described in the section 4.1.2.1.

The callback methodology described in the section 3.3 was fundamental to achieve a good communication system between the different modules. Moreover, this system is strong enough to work with future additions. This communication was essential to the development of the streaming and prediction features described in the section 4.1.2.1. This is because an online feature is required for a future work.

In order to develop this application, the following requirements were established:

4.1.1 Application requirements

The application requirements are divided into functional requirements and non-functional requirements:

Functional requirements:

- Online data processing
- Plotting the online input data with different channels, for any BCI input (e.g. OpenBCI and offline data)
- A simple train and predict system to test the feature
- Ability to save the input data in a SQL DB
- Simple classification system (for future learning machines systems)

Non-functional requirements:

- Interaction with different BCI data sources (e.g. OpenBCI and offline data)
- Asynchronous communication with the hardware
- Online plotting system
- Log with application progress
- Different threads for the BCI control software and EEG plot system
- Busy fault tolerance
- Modular implementation
- Ability to Work with external offline and online applications

4.1.2 Design of the main application

The main application was divided into 5 modules (see figure 4.1), one of them coordinating the rest. The main module should be used as a communication pipe between the others. Indeed, this could represent a bottleneck for the periodic tasks. Hence, the communication tasks were oriented to communicate the modules without sharing the data across the main module (if possible). The I/O modules (BCI and data storage) are designed for a driver connection system, due to the need of using different external systems with it in a future work.

Once the process handler has launched the BCI thread, the different streaming channels appear. This system has a plotting frequency determined in the creation of the module. It plots one row for each activated channel. Additionally, a common operating system (without any kernel patch) can't be used with *real time* processes. The limitations of

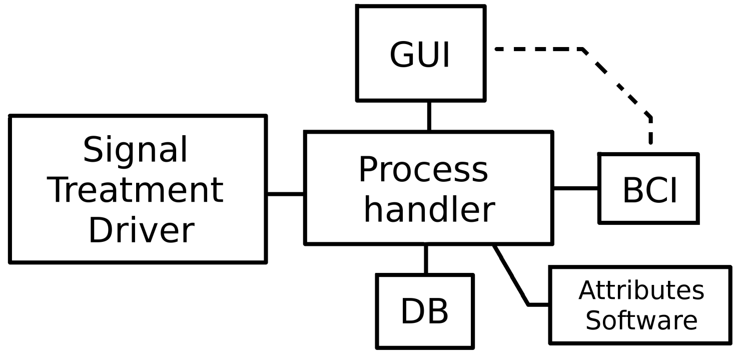

Figure 4.1: Scheme of the design. The noncontinuous line is a connection to accelerate the data communication (bottleneck)

the GUI system are described in section 4.1.3.6. However, the real time feature is not necessary (section 4.1.3.6), but it can be implemented in the future with the use of an RTAI (Real Time Application Interface for Linux).

4.1.2.1 Process handler

This is the main module, which handles the flow of the application. It manages the communication with the others systems and launches the two main features of the application for this project: simple data stream and train and predict stream system. Both systems save the recorded data in the database. This is important for future analysis.

Simple data stream: The first functionality of the application is to launch a simple GUI. It provides an online stream of data, taken from the defined BCI.

The GUI shows the EEG signal taken for each selected sensor (Figure 4.2). This representation is useful to check the correct connection of the BCI system, in particular, to check the position of each electrode. The flow of this functionality is described in figure 4.3.

Train and Predict stream: This system is a first contact with BCI train/prediction systems. The training system works like the simple data stream with two differences. Each time the system sends a signal data package from the BCI, the process handler module asks the application for the attribute information. The second difference is the target attribute. The target is specified by the application in the training phases. The training should alternate between different attributes, with more than one repetition for each attribute.

Once the application has built the learning machine, the system starts again. In this case the subject should look at him/her desire attributes. The application will repeat the attributes to average the EEG signal of each one. Finally the application will show the detected attribute.

This feature was only tested with the offline BCI, which has a high sample frequency. The execution time was really slow, hence needs improvement in future work.

The flow of this functionality is described in figure 4.4.

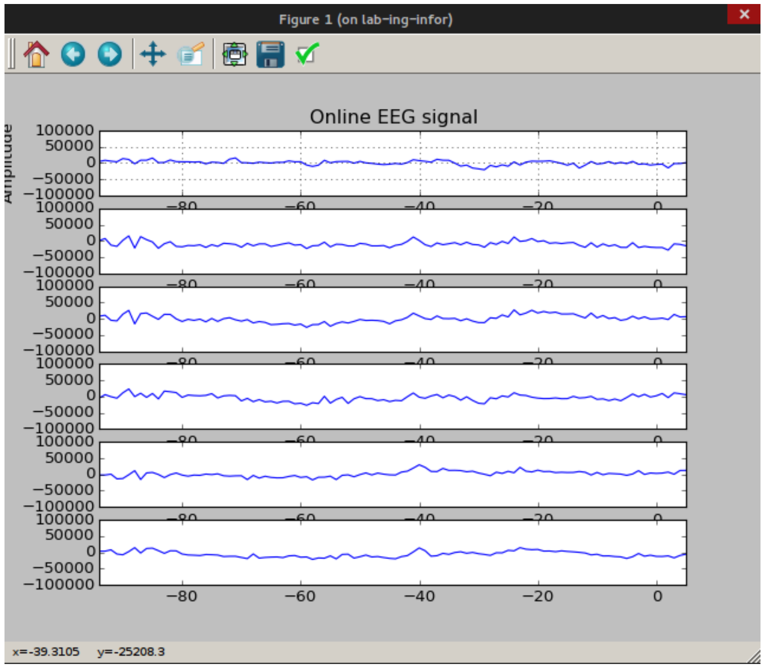

Figure 4.2: Screen-shoot of the GUI interface for offline data

4.1.3 Description of usage

For the first version of the application, the detect functionality only shows the detected attribute, but in future work the application will be able to communicate with other systems (e.g. export the spelling prediction to a chat software). Moreover, this feature is only a test, hence the not good enough success rate.

The software has the following modules:

4.1.3.1 BCI driver

The communication system works under a callback flow, which provides a simple method to add different BCIs (SDK to communicate with the hardware or offline data). The BCI implements the functions related to the data stream and training stream. Both work under a callback (implemented in the process handler). With this system, the streaming is launched in a different thread, followed by the data classification.

Thus, the module needed for the communication with a BCI is simple. The application has an offline and an online driver for two different BCIs. The offline driver is designed for the competition data mentioned in section 3.1, while the online driver was developed to work with the open source BCI *OpenBCI* mentioned in section 2.2.2.3. Both drivers were tested for the simple stream feature with good results. In the case of the OpenBCI, the Python SDK (section 4.1.3.1) has a stream callback function.

Offline BCI: The offline driver streams the stored data with an accorded frequency. However, this application was developed to parse the competition data described in section 3.1.1. The GUI stream works correctly.

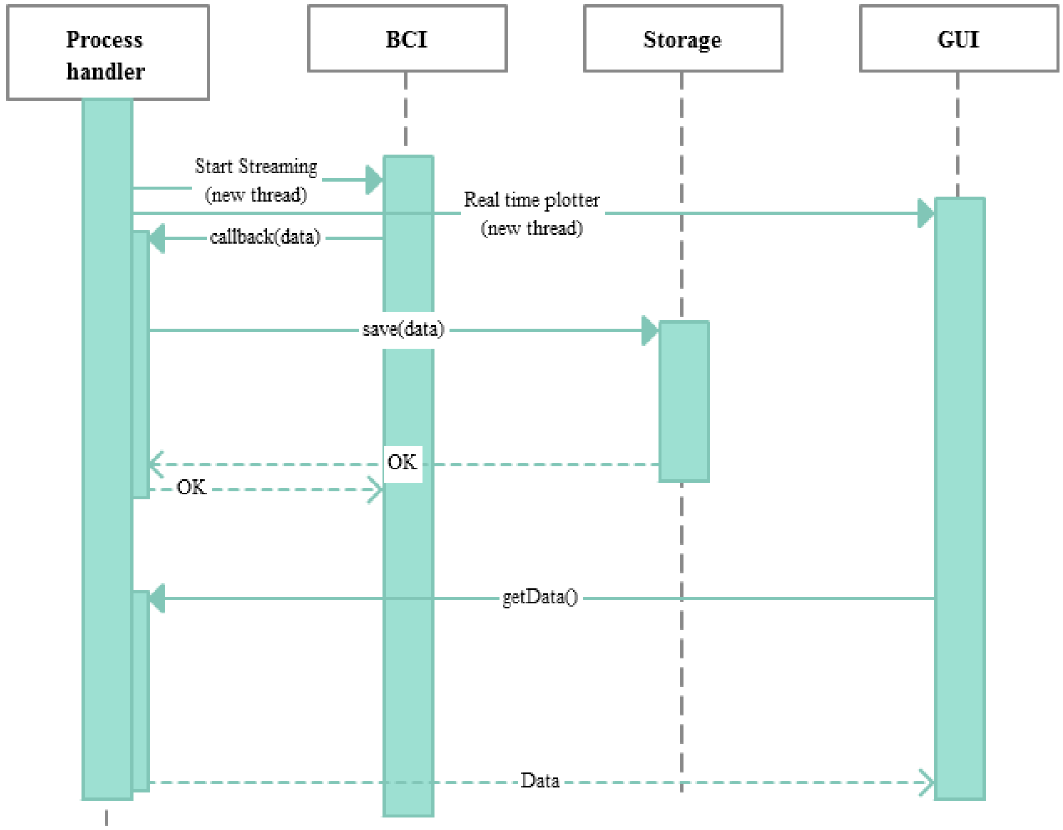

Figure 4.3: Sequence diagram of the simple data stream feature. The process handler begins the communication, launch the BCI and GUI threads. The BCI thread invokes the process handler callback to send the new data to it. The process handler returns an OK after storing the data in the storage module. On the other thread, the GUI asks for new information continuously

OpenBCI: OpenBCI has an Open SDK in continuous development in Python to communicate with the hardware platform. The streaming function is already implemented in the SDK, so it doesn't need an external BCI driver. Once the data go through the callback, the application should work independently of the data source. On the other hand, the OpenBCI software has an alternative communication system, which works under software demand.

The OpenBCI was connected with the GUI system and it works without problems. The connection needed is described in the appendix 4.4. At the end of the project the OpenBCI was broken, and the time needed to repair it was too long.

4.1.3.2 Storage

This module saves the data taken from any source into a database (MySQL), with previously configured parameters. The default configuration is the one which works in the laboratories. The data insertion was tested for the GUI stream and predict stream. This feature was implemented using the pymysql library to connect with the database.

4.1.3.3 Signal Treatment

The Signal treatment integrates the system described in section 4.1.4.3 and uses the driver methodology described in section 3.4. Additionally, it classifies the signals in a structure. Once the process handler requests a prediction, the data is used to predict, pondering each window's individual prediction and the average window prediction. Afterwards, the structure is deleted. As is described in section 1.1, this system is an introduction to make a real detection system. It implements common signal functions (average, bandpass, normalized, windowed) and it includes more complex libraries to make other process (SVM described in section 3.2.2.1). However, the system described in [14] defines a more complex system with different SVM systems and more precise filter values (3.1.1).

4.1.3.4 User Application

For a correct work flow of the predict system, the software needs an application that provides the information related with the recorded signal. This information is the attribute shown, even if there is no attribute shown (with no attribute shown, the application has to send a "None" attribute). Moreover, if the prediction system asks for the target attribute, the application returns '1' if the target character is being shown, and '0' otherwise.

This design provides a transparent treatment of the data, making it go through the process handler. This provides a simple connection with other applications (e.g. a row of pictures).

For this project, a simple application was designed only to test the developed functionality.

Dummy application: This application was designed to work like a test that always shows the same character, not knowing if it is desired or not. However, this is only a test application.

4.1.3.5 GUI

In some cases, a visual representation of the EEG signal can be used to discover its properties. Additionally, it can be used for a subsequent classification or even to check the contact of the electrodes. The GUI module is designed to solve these needs, with a system which receives the signal from any source to plot it (offline or online). For this project the GUI module only implements a simple data stream to view the signal taken from the BCI.

4.1.3.6 Considerations and limitations of the Software

This software has some limitations due to the complexity the following tasks:

Synchronization BCI-Software: The BCI hardware works under a frequency given through an argument. For that reason, the GUI module has a initialization value to configure it, due to it running in a single thread. Other tasks work under hardware demand. Thus, when an instance of data is called from the BCI, the system can't take

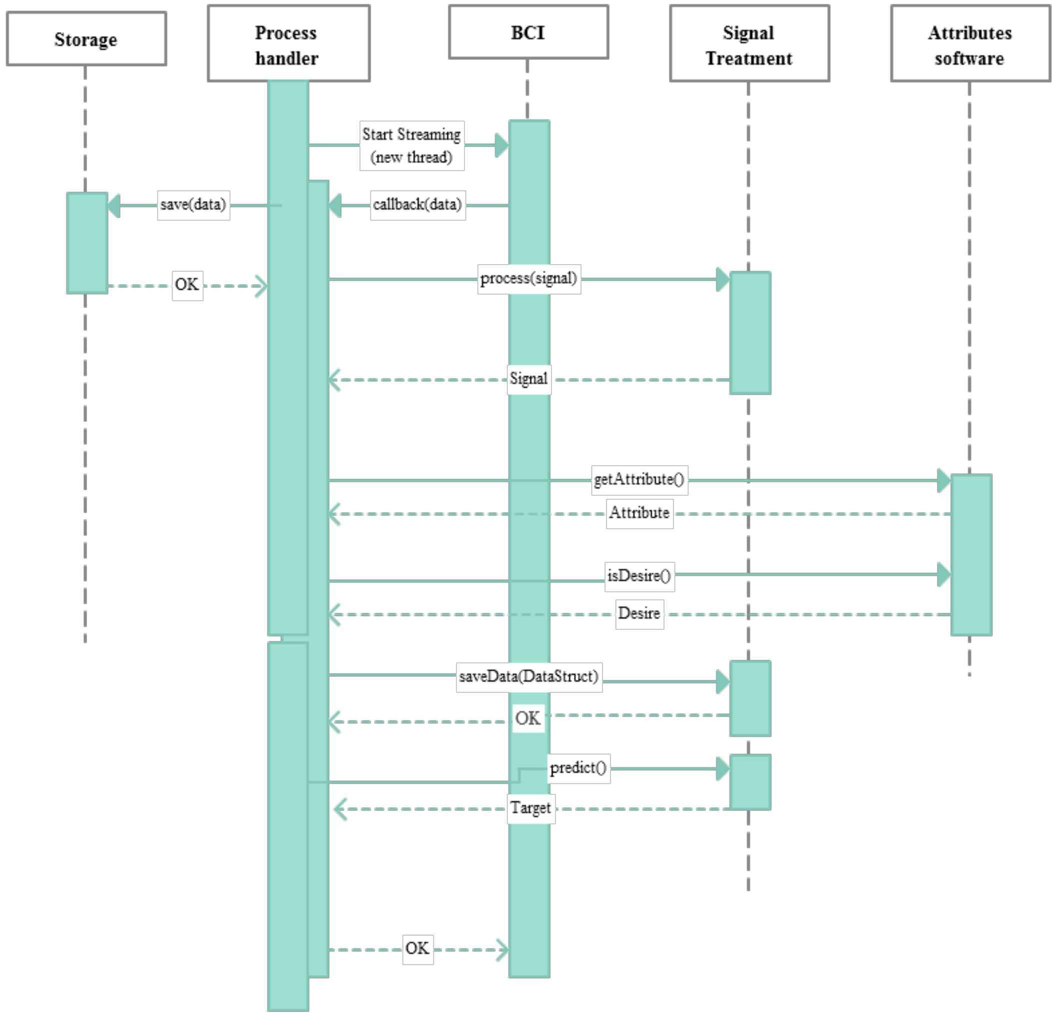

Figure 4.4: Sequence diagram of the train and predict functionality. The process handler begins the communication, launching the BCI training thread. The BCI thread invokes the process handler callback to send it the new data. The process handler stores the data in the storage module. Once the storage is over, the data is sent to the signal treatment module. This module stores it in a classified structure after being filtered. If the program is in training phase, the attributes software (application described in section 4.1.3.4) provides data related to the signal. In the test phase, the signal treatment will return a score array, which is interpreted by the attribute software

new data until the signal is be classified (see section 4.1.2.1). The GUI update works under a timer, so it might not be executed in time (O.S. scheduling).

O.S. scheduling: Any process which works under a common O.S. can't be run in real time due to the process scheduling of the O.S. (although there are some kernel patches to execute real time process, as is described in section 4.1.2). Hence, the *real time tasks* should be called *online tasks*.

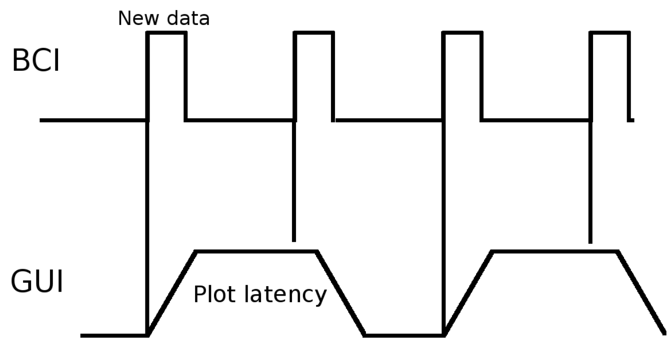

Figure 4.5: Scheme of the sync problems between the GUI and the BCI modules. If the frequency received data exceeds the latency generated by the processing task, thanks to the callback functionality, the application only ignores this data. However, this data is always saved in the database module (as is described in section 4.1.3.2)

Noisy signal: The data from the Competition was very noisy, due to the short period of time between events and the P300's difficulty. This makes the signal harder to classify. Indeed, the solution taken from the competition reduces the problem, but it doesn't solve it. Although the winner method had a 0.97 success rate, the configuration of the SVM and the appropriate signal filters weren't easy tasks (see section 3.2.1). Hence, the results of the prediction systems are not as good as the competition ones.

Others: The software was tested with the Offline BCI platform for the most part of time, due to the late arrival of the OpenBCI platform. For that reason, the train and prediction system is not tested in the OpenBCI platform. Moreover, the OpenBCI was broken, so the final test couldn't be executed with it.

4.1.4 Application results

The application resulted of this end degree project fulfills the objectives established in section 1.1. However, the accomplishment of this objectives requires testing results to attest it. Thus, the following features were tested: driver functionality, callback method and a simple classification process.

4.1.4.1 Driver tests

The driver configuration is needed to test the modular design of the project. It was tested for the three different drivers: Application, BCI and signal treatment driver.

Application: For this test, two different applications were developed: an Offline application and a dummy application. The first application provides the shown attribute data, and, in the training phase, if it was desired or not. The second always returns the same attribute with no desired attribute information. The project works correctly with both applications.

BCI: In the BCI test, the offline BCI and OpenBCI SDK were tested only for the GUI feature, due to the OpenBCI hardware being broken. However, the offline BCI was tested in the predict feature. In the first test, the results were the sames for both platforms. This means that the driver functionality works correctly, because once the data is processed by the process handler module the source doesn't mind.

Signal treatment driver: This module was tested with different classification systems, implemented in copies of it. The tests were executed without problems with the different systems.

4.1.4.2 Callback method

The callback method is necessary to check the fault tolerance of the software. To test this method, the following features were executed:

GUI: In order to test this method, the GUI feature was executed with different frequencies. This produces inferior plot resolution in the GUI (according to section 4.1.1). Nevertheless, this tests were enough to test the endurance of the callback method.

Offline simulations:

Train and predict system: The objective of these tests was to check the no data was lost. After some tests of the offline BCI, the application shows no difference in the resulting data. Additionally, the processing task doesn't slow down enough to create a delay. This is due to the low sampling frequency ($\lesssim$ 500Hz).

4.1.4.3 Simple classification system

The last feature specified in the section 4.1.1 was a simple classification system. The objective of this system is to create a functional environment and a simple prediction system. The functional environment is important, in order to work in the future with this tool, due to the complexity of adding new features to the environment. Additionally, a simple prediction system was developed as a first contact with the machine learning systems needed for the future application. The chosen system was the following SVM:

Prediction system: The SVM was configured with values of C in the interval $[10^{-2}, 10^{10}]$ and values of γ in $[10^{-}9, 10^{3}]$. The signal data was bandpassed between 0.1Hz and 10Hz. To train the SVM, the signal was averaged per row and column, and classified into ERP and non-ERP. In order to reduce the simulation system cost, the train percent was fixed at 80%, with a 5-cross validation.

For 85 character epochs, the SVM was trained and obtained a 30% raw test error over the average signal. This error percentage is classified as "raw", due to the individual classification not being enough to test the system. A prediction needs to achieve row and column separately. For that reason, a vote function was developed, to get more accurate prediction results. This system combines the results of the continuous classifications of the system and the classification of the average.

This function gives the percentages of error for a top system and a best system. The best system only considers the row and column with the best vote value, while the top system returns a set of possible values. This second system is designed to work with other classification systems in order to improve their detection rate. It was tested with a simple peek algorithm, and the intersection of both sets. This peek algorithm was developed to get the score of an averaged interval, which should contain the ERP. Additionally, the average of the two external intervals was subtracted from them. This interval is calculated from a barrier search algorithm applied to the P300 intervals of the data set. In the competition data description a similar algorithm is defined as a test algorithm to detect the ERP.

4.2 Conclusions

This end degree project was created as an introduction to the BCI world. Additionally, it will be used for a future work in the area in a master's program (Master's Degree in ICT Research and Innovation). The initial knowledge when facing this project was limited. Some knowledge like the study of a signal or the different filters to use in it (e.g. Butterworth, Chebyshev) were acquired during the development of the project. However, the machine learning system knowledge was limited too, and to deal with it, an .txt to .arff script was generated at the beginning of the project. Thus, the initial data generated could be classified with the Weka tool, but it was replaced by the scikit learn library, which was more useful, due to the whole project being developed in Python. This programming language was chosen due to the amount of scientific libraries available and for its simple coding interface.

As is described in section 3.2.1, the P300 control signal is a complex pattern. This is due to the fact that the P300 does not necessarily manifest when the subject perceives a column/row intensification. For that reason, the prediction system was developed as an introduction to better classification systems.

In order to develop a system with the requirements described in section 4.1.1, the callback and driver methods were included into the design of the project. The transparency of the communication used in these methods is very high. The callback functionality provides abstract methods, asynchronous communication and reusable code.

Finally, the development of a train and classification system for the offline data shows a deficit in the features of the P300 implementations. Each person shows individual patterns that modify the score result of one particular configuration. For this reason, the design of individual BCI solutions for individual problems is not the best way to face the BCI problem. On the other hand, the world of UI has grown from the old systems, which needed a complex configuration from the user, to systems that configure themselves for each user. Thus, the BCI world should develop closed systems that could adapt themselves for each user, with no configuration. Just like the system described in [16], which can adjust the detection in a *closed loop* configuration for the SSVEPs control signal. This systems are only possible with online signal process, which increments the complexity.

4.3 Learned skills in this project

The BCI area is a multidisciplinary research theme. In the development of this end of degree project, the following knowledges areas were used:

State of the art
: The development of a BCI system needs a detailed study of the different BCI based options. Although some BCI options are not accessible due of their high cost, a knowledge of different BCI systems can be useful to find a good solution. Knowing the control signals limits and common resolution gives an idea of which one is more viable for a particular problem. These, along with the different BCI features described in section 2, are essential for any BCI project.

Callback and driver methodology
: These methods were helpful to avoid the problems of online signal processing and modular systems as is described in sections 3.3 and 3.4.

Hardware online connection
: The online communication with external hardware needs to establish a middle communication point between the hardware SDK and the software application.

Signal filters
: The signal was filtered between many frequencies, with a simple Fourier band pass filter between 0.1Hz and 10Hz. Additionally, another feature extraction and selection methods were tested, as is described in sections 3.2.2 and 3.2.2.

Study of EEG behavior
: In order to improve the accuracy of the prediction system, a dual prediction system was studied. The idea of this system was the use of a second classification system. It should study different features for a final discrimination between both systems. For this purpose, a simple peek algorithm was developed. This system scored each event like the average of the *peek interval* minus the average of the previous and next intervals (described in 4.1.4.3).

Variables simulation
: The learning machine tools provided from the scikit python library were useful but improvable. The appropriate configurations for the different SVMs were implemented with the simulation modules, but the iterations were inefficient in some levels. Once the iterations were over, the simulation needed to create another SVM for the new parameters. Additionally, some variables were fixed to reduce the simulation costs.

: A variable simulation system inside of the SVM library would had been useful in order to reduce the costs of an external and inefficient simulation.

Machine learning and features selector and extractor methods
: The prediction system included in the project was chosen among others in order to find good results. One example was a SCP bandpassed system with 2 filters and a SVM to classify variance of the resulting events. This system shows an excellent

learning, but fails with the test set. Another example was the study of Principal Component Analysis (PCA) described in section 3.2.2.

Others

The new programming language (Python) or handing an embedded system (OpenBCI hardware) also added difficulty to the project.

4.4 Future work

Some ideas to improve this project are:

- The development of a true detection system to achieve a full detection for the offline data and other sources.
- The development of a meta-learning system which could find, with some tools, patterns in the behavior of each subject, for better classification rates and less training times.
- The replacement of the multi-thread system for a multicore one, in order to achieve a better response time.
- The addition of more code drivers for other BCI platforms to compare the results of each algorithm in different BCIs.
- A SSVEPs detection system, and an interface to select the detection to be processed.
- The improvement of the train and predict system.
- The use of a real time patch for GNU/Linux O.S. in order to compare the real time tasks and online tasks.

Appendix A

Code application

A.1 Process handler

```

__author__ = 'Guillermo Sarasa Duran'
from time import sleep
import logging as log
import datetime as dtm
from threading import Thread
import GUI.GUI_functions as guis
#import Storage.DB as db
log.basicConfig(filename='example.log', level=log.DEBUG)

class BCI_kernel:

    bci_system = []
    std_conf = []
    storage_system = []
    stream_thrd = []
    training_thrd = []
    called = 0
    sample=0

    eeg_str = []
    gui_eeg = None
    def __init__(self, bci_system, std_conf, storage_system,
      application, channels=6):
        self.sens = channels
        self.bci_system = bci_system
        self.std_conf = std_conf
        self.storage_system = storage_system
        self.application = application
        self.gui_eeg = guis.gui_sampler(channels)
```

```
        try:
            self.slct_sns = application.selected_sensors
        except:
            self.slct_sns = -1

    def log_error(self, msg):
        log.error(self.time() + ":[BCI_Kernel]: " + msg)

    def log_info(self, msg):
        log.info(self.time() + ":[BCI_Kernel]: " + msg)

    def time(self):
        return dtm.datetime.now().time().isoformat()
    #def classification_system(self, ):

    def train_predict_stream(self):
        self.log_info("Starting application interface...")
        if self.application.start() == -1:
            self.log_error("ERROR: Can't start application, exiting...")
            return
        self.log_info("BCI successfully launched")

        self.log_info("Starting DB")
        self.storage_system.connect()
        self.storage_system.close()
        self.log_info("DB successfully launched")


        self.log_info("Channels selected")
        self.log_info("Stating training...")
        self.launch_training_streamer()
        self.log_info("Training complete")

        self.log_info("Starting BCI stream with the selected configuration...")
        self.launch_data_streamer()
        if self.check_connection() == -1:
            print "Can't connect to BCI"
            return

        self.console_interface()

        self.log_info("Closing stream...")
        #Halt BCI
        self.log_info("Closed")
```

```


    def online_stream_signal(self):
        self.log_info("Starting application interface...")
        if self.application.start() == -1:
            self.log_error("ERROR: Can't start application, exiting...")
            return
        self.log_info("BCI successfully launched")

        self.log_info("Starting BCI stream with the selected configuration...")
        #self.bci_system.print_register_settings()
        self.launch_gui_streamer()
        if self.check_connection() == -1:
            print "Can't connect to BCI"
            #return

        self.stream_gui()

        self.log_info("Closing stream...")
        #Halt BCI
        self.log_info("Closed")

    def stream_gui(self):
        self.gui_eeg.real_time_plotter(self.getEEG_simple)
        self.console_interface()

    def console_interface(self):
        string = raw_input("Press [X] to kill the application")
        while string != 'X':
            string = raw_input("Press [X] to kill the application")
        return

    def select_channels(self):
        for chnl in raw_input("Please, introduce the desire channels separating it with ',' \n").split(' '):
            if self.bci_system.set_Channel(chnl, 1) == -1:
                print "The channel " + str(chnl) + " doesn't exist, ignoring"
                self.log_error("The channel " + str(chnl) + " doesn't exist, ignoring")


```

```
    def launch_training_streamer(self):

        self.us = self.storage_system.subjct
        self.sess = self.storage_system.session

        self.training_thrd = Thread(target=self.bci_system.start_training_stream, args=((self.training_callback,self.std_conf.learn)))
        self.training_thrd.start()
        self.training_thrd.join()



    def training_callback(self, stream_data):
        self.called = 1
        attribute = self.application.getAttibute()
        knw = self.application.getDesired()
        data_matrix = [[] for i in range(stream_data.__len__())]
        for sens in range(0, stream_data.__len__()):
            data_matrix[sens] = stream_data[sens]
            if self.slct_sns != -1:
                senss = self.slct_sns[sens]
            else:
                senss = sens
            self.storage_system.insert( "INSERT INTO signalData (idUser,idSession,Sensor,Sample,Data) values (" + str(self.us) +"," + str(self.sess) + "," + str(senss) +"," + str(self.sample)  + "," + str(stream_data[sens])  + ");")


        self.std_conf.training({ 'Attribute' : attribute , 'Dsr' : knw , 'Data' : data_matrix })
        self.sample +=1
        return

    def getEEG_simple(self):

        return self.eeg_str

    def launch_data_streamer(self):
        self.std_conf.reset()
        self.us = self.storage_system.subjct
        self.sess = self.storage_system.session
        self.log_info("Starting prediction stream...")
```

```
        self.stream_thrd = Thread(target=self.bci_system.start_streaming, args=([self.callback]))
        self.stream_thrd.start()
        self.log_info("Prediction␣stream␣launched")
    def launch_gui_streamer(self):
        self.us = self.storage_system.subjct
        self.sess = self.storage_system.session
        self.log_info("Starting␣GUI␣stream...")
        self.stream_thrd = Thread(target=self.bci_system.start_streaming, args=([self.simple_callback]))
        self.stream_thrd.start()
        self.log_info("GUI␣stream␣launched")


    def simple_callback(self, stream_data):
        try:
            self.eeg_str = stream_data.channel_data[:]
            for sens in range(0, stream_data.channel_data.__len__()):
                if self.slct_sns != -1:
                    senss = self.slct_sns[sens]
                else:
                    senss = sens
                self.storage_system.insert( "INSERT␣INTO␣signalData␣(idUser,idSession,Sensor,Sample,Data)␣values␣(" + str(self.us) +"," + str(self.sess) + "," + str(senss) +"," + str(self.sample)  +"," + str(stream_data[sens]) + ");")

        except:
            self.eeg_str = stream_data[:]
            for sens in range(0, stream_data.__len__()):
                if self.slct_sns != -1:
                    senss = self.slct_sns[sens]
                else:
                    senss = sens
                self.storage_system.insert( "INSERT␣INTO␣signalData␣(idUser,idSession,Sensor,Sample,Data)␣values␣(" + str(self.us) +"," + str(self.sess) + "," + str(senss) +"," + str(self.sample)  +"," + str(stream_data[sens]) + ");")

        self.sample+=1
        return self.eeg_str
```

```

    def callback(self, stream_data):
        self.called = 1
        attribute = self.application.getAttibuteTest()
        knw = self.application.getDesiredTest()
        data_matrix = [[] for i in range(stream_data.__len__())]
        for sens in range(0, stream_data.__len__()):
            data_matrix[sens] = stream_data[sens]
            if self.slct_sns != -1:
                senss = self.slct_sns[sens]
            else:
                senss = sens
            self.storage_system.insert( "INSERT INTO signalData (idUser,idSession,Sensor,Sample,Data) values (" + str(self.us) +"," + str(self.sess) + "," + str(senss) +"," + str(self.sample)  + "," + str(stream_data[sens])  + ");")

        res = self.std_conf.classify({ 'Attribute' : attribute , 'Dsr' : knw , 'Data' : data_matrix })
        if res != 0:
            self.application.predict()
        self.sample+=1
        return

    def check_connection(self):
        self.log_info("Checking connection...")
        for i in range(5):
            sleep(1)
            if self.called == 1:
                self.log_info("Connected")
                return 1

            self.log_info("Error, Retrying connection to BCI system...")
        self.log_error("Error, can't connect to BCI system")
        return -1
```

A.2 Dummy application

```
__author__ = 'Guillermo Sarasa Duran'
import numpy as np

```

```
class dummy:


    def __init__(self):
        a = 1+1

    def getAttibute(self):
        return 'A'

    def start(self):
        print "YEY"
        return 1

    def getDesired(self):
        return -1


    def time(self):
        return dtm.datetime.now().time().isoformat()
```

A.3 Offline BCI

```

__author__ = 'Guillermo␣Sarasa␣Duran'
import re
import logging as log
import shutil as shl
import datetime as dtm
import numpy as np
import time as tmp

def print_hello():
    print "hola"

class OfflineBCI:
    f = -1
    name = "OfflineBCI"
    file = "example.txt"

    route = "/home/guillers/TFG/TFG/Files/"

    file_1 = "Subject_A_Train_Signal.txt"
    raw_matrix = []
    file_2 = "Subject_A_Test_Signal.txt"
```

```
    test_raw_signal = []


    num_sens = 20
    trl = 1
    len = -1
    hz = 0
    latency = 0
    delaySum = 0
    seek = 0
    seekFlag = 0
    data_signal = []
    test_signal = []
    selected_sensors = ()
    max_trl = 0
    actual_time = 0
    sync = 0
    raw_matrix = None
    test_raw_signal = None

    def __init__(self, route, selected_sensors,num_sens=64, sync=[[0,24,42],[1,0]],  length=7794, trial=0, max_trials = 85, hz=240, delay_emulation=2):
        self.log_info("BCI initializing...")
        self.sync = sync
        self.selected_sensors = selected_sensors
        self.latency = delay_emulation
        self.delaySum = 0
        self.max_trl = max_trials
        self.len = length
        self.freq = hz
        self.trl = trial
        self.num_sens = num_sens
        if (max_trials < 1) | (trial > max_trials) | (hz > 240) | (length != 7794) | (selected_sensors.__len__() > 64):
            self.log_error("Wrong parameters, try with the default configuration")
            return -1

        self.raw_matrix = np.genfromtxt(route + self.file_1)

        self.test_raw_signal = np.genfromtxt(route + self.file_2)

        self.log_info("BCI initializing success")
```

```

    def start(self):
        """Start␣the␣current␣BCI"""
        self.log_info("BCI␣starting...")

        self.log_info("Files␣loaded")
        self.log_info("Generating␣matrix...")

        self.data_signal = np.empty((85,64,7794))
        for i in range(0, 85):
            self.data_signal[i] = self.raw_matrix[0+(64*i):64+(64*i)]

        self.test_signal = np.empty((100,64,7794))
        for i in range(0, 100):
            self.test_signal[i] = self.test_raw_signal[0+(64*i):64+(64*i)]

        self.log_info("Matrix␣successfuly␣generated")
        self.log_info("BCI␣starting␣successfuly")
        return 0


    def start_test_stream(self, callback, predict):
        self.log_info("Start␣test")
        for streamed_trial in range(0, 100):
            self.log_info("Progress:␣" + str(streamed_trial))
            for time in range(0, self.len):
                strm_data = []
                for sens in range(0, self.test_signal[streamed_trial].__len__()):
                    if sens in self.selected_sensors:
                        strm_data.append(self.test_signal[streamed_trial][sens][time])
                callback(strm_data)
            predict()
        self.log_info("End␣of␣test")

        return 0

    def start_streaming(self, callback):
        self.log_info("Streming␣initiated")
        for streamed_trial in range(0, self.max_trl):
            for time in range(0, self.len):
                strm_data = []
```

```
                for sens in range(0, self.data_signal[streamed_trial].__len__()):
                    if sens in self.selected_sensors:
                        strm_data.append(self.data_signal[streamed_trial][sens][time]*1000)
                callback(strm_data)
                tmp.sleep(0.0041666666666666666)
        self.log_info("End of the data stream")
        return 0

    def start_training_stream(self, callback, learn):
        self.log_info("Start training")
        for streamed_trial in range(0, self.max_trl):
            self.log_info("Progress: " + str(streamed_trial/float(self.max_trl)))
            for time in range(0, self.len):
                strm_data = []
                for sens in range(0, self.test_signal[streamed_trial].__len__()):
                    if sens in self.selected_sensors:
                        strm_data.append(self.data_signal[streamed_trial][sens][time])
                        self.log_info("trial: " + str(streamed_trial) + " time: " + str(time) + " sens: " + str(sens))
                callback(strm_data)
            learn()
        self.log_info("End of training")

        return 0



    def time(self):
        return dtm.datetime.now().time().isoformat()

    def log_error(self, msg):
        log.error(self.time() + ":[BCI->Offline]: " + msg)

    def log_info(self, msg):
        log.info(self.time() + ":[BCI->Offline]: " + msg)
```

A.4 Signal Treatment Driver

```

__author__ = 'Guillermo Sarasa Duran'
import logging as log
log.basicConfig(filename='example.log', level=log.DEBUG)
import numpy.fft as fourier
import numpy as np
import datetime as dtm
from matplotlib.mlab import PCA
import matplotlib.pyplot as plt
import math as math

from scipy.linalg import svd
from scipy.misc import lena
from sklearn import svm

ERR = -1

class signal:
    #Width of the window
    width = 0

    #var to config the resolution
    conf_res = 0.0

    #Base resolution for all methods
    base_res = 0.0

    #Value to start
    conf_res = 0  # conf_res == 2^n
    base_res = 0  # base_res == 2^n
    initialize = 0

    #the quantity of
    sens_number = 20

    swap=0

    LM = []

    dt = 0
    c = 1
    Gamma = 1
    train = 0.8

    Atrbt = -1
    Dsr = -1
```

```

    cont = -1

    time = 0

    DATA_STRUCT = {}
    TIME_STRUCT = {}
    FREQ_STRUCT = {}
    FREQ_DETECTION_ARRAY = {}
    scr_sml = 0
    scr_big = 0
    t_simple = 0
    yes = []
    no = []



    def __init__(self, width, conf_res, base_res, sens_number, scr_sml, scr_big, C=1.152, Gamma=1.408, train=0.8, ini=0, end=100):
        self.c = C
        self.scr_sml = scr_sml
        self.scr_big = scr_big
        self.ini = ini
        self.end = end
        self.cont = 0
        self.Gamma = Gamma
        self.train = train
        log.info(dtm.datetime.now().time().isoformat() + ":[STD->driver]: Initializing")
        self.sens_number = sens_number
        self.width = width
        self.conf_res = conf_res
        self.base_res = base_res
        if conf_res > base_res:
            log.CRITICAL(dtm.datetime.now().time().isoformat() + ":[STD->driver]: ERROR, the config resolution can't be bigger than the min")
            return
        self.initialize = 1
        log.info(dtm.datetime.now().time().isoformat() + ":[STD->driver:]: Initializing success")
        return

    def reset(self):
        self.DATA_STRUCT = {}
```

```
        self.TIME_STRUCT = {}
        self.FREQ_STRUCT = {}
        self.FREQ_DETECTION_ARRAY = {}


    def learn(self):
        for indx in self.DATA_STRUCT:
            line = []
            for sens in self.DATA_STRUCT[indx]['Data']:
                line.append(np.average(sens, axis=0)[self.ini:self.end][:]) #I hope it works
            if self.DATA_STRUCT[indx]['Dsr'] == 1:
                self.yes.append(line)
            else:
                self.no.append(line)
        self.DATA_STRUCT = []

    def SVM_generator(self, training_yes, training_no ):
        train_num = int(training_yes.__len__()*self.train)
        self.log_info("Generating SVM machine...")
        train_data = [[] for i in range (train_num*2)]
        train_data[:train_num] = training_yes[:train_num][:]
        train_data[train_num:] = training_no[:train_num][:]
        score = [0 for i in range(train_num*2)]
        for i in range(0, train_num): score[i] = 1

        self.LM = svm.SVC(C=self.c, gamma=self.Gamma)

        self.LM.fit(train_data, score)
        self.log_info("SVM machine succesfully generated")

        return self.LM

    def save_data(self, bci_data, Atrbt, Dsr,freq,time):
        if freq == -1: return 0
        if self.DATA_STRUCT.has_key(str(Atrbt)) == False:
            self.DATA_STRUCT.update({str(Atrbt) : {'Dsr' : Dsr , 'Data' : [[] for i in range(bci_data['Data'].__len__())]}})
        for sens in range(0, bci_data['Data'].__len__()):
            if freq+1 > self.DATA_STRUCT[str(Atrbt)]['Data'][sens].__len__():
                self.DATA_STRUCT[str(Atrbt)]['Data'][sens].append([0 for k in range(240)])

```

```
                self.DATA_STRUCT[str(Atrbt)]['Data'][sens][freq][time] = bci_data['Data'][sens]

        return 0

    def predict(self, indx):
        if self.LM == []:
            self.SVM_generator(self.yes, self.no)
        detc = 0
        simple_score = 0
        for sens in self.DATA_STRUCT[indx]['Data']:
            for time in sens:
                simple_score += self.LM.predict(time)
            if self.LM.predict(np.average(sens, axis=0)) == 1:
                detc += 1
        if detc >= self.DATA_STRUCT[indx]['Data'].__len__()/2:
            if self.FREQ_DETECTION_ARRAY.has_key(indx) == True:
                self.FREQ_DETECTION_ARRAY[indx] += scr_big + (simple_score * self.scr_sml)
            else:
                self.FREQ_DETECTION_ARRAY.update({ indx : scr_big + (simple_score * self.scr_sml)})

    def classify(self, bci_data):

        Atrbt_aux = bci_data['Attribute']
        Dsr_aux = bci_data['Dsr']

        if self.Atrbt != Atrbt_aux:
            self.TIME_STRUCT.update({str(Atrbt_aux) : 240})
            if self.FREQ_STRUCT.has_key(str(Atrbt_aux)) == False:
                self.FREQ_STRUCT.update({str(Atrbt_aux) : 1})
            else:
                self.FREQ_STRUCT[str(Atrbt_aux)] += 1
            self.Atrbt = Atrbt_aux
        indx = ''
        indx_lst = [indx for indx in self.TIME_STRUCT]
        for indx in indx_lst:
            cont = self.TIME_STRUCT[str(indx)]
            if cont == 0:
                del self.TIME_STRUCT[str(indx)]
                continue
```

```
            else:
                self.TIME_STRUCT[str(indx)] -= 1
            self.save_data(bci_data, indx, -1,self.FREQ_STRUCT[str(Atrbt_aux)]-1,239-self.TIME_STRUCT[str(indx)])
        if self.swap >= 235:
            for index in self.DATA_STRUCT:
                self.predict(DATA_STRUCT[str(index)])
            best_score = -1
            best_strg = '.'
            best_strg1 = '.'
            best_strg2 = '.'
            cont = 11
            return np.sort(self.FREQ_DETECTION_ARRAY.items())

        return 0

    def training(self, bci_data):
        Atrbt_aux = bci_data['Attribute']
        if Atrbt_aux == None:
            self.swap += 1
        else:
            self.swap = 0

        Dsr_aux = bci_data['Dsr']

        if self.Atrbt != Atrbt_aux:
            self.TIME_STRUCT.update({str(Atrbt_aux) : 240})
            if self.FREQ_STRUCT.has_key(Atrbt_aux) == False:
                self.FREQ_STRUCT.update({str(Atrbt_aux) : 1})
            else:
                self.FREQ_STRUCT[str(Atrbt_aux)] += 1
            self.Atrbt = Atrbt_aux
        indx_lst = [indx for indx in self.TIME_STRUCT]
        for indx in indx_lst:
            cont = self.TIME_STRUCT[str(indx)]
            if cont == 0:
                #self.predict(str(indx))
                del self.TIME_STRUCT[str(indx)]
                self.FREQ_STRUCT
                continue
            else:
                self.TIME_STRUCT[str(indx)] -= 1
            if indx == Atrbt_aux:
                self.save_data(bci_data, indx, Dsr_aux,self.FREQ_STRUCT[str(Atrbt_aux)]-1,239-self.
```

```
                  TIME_STRUCT[str(indx)])
            else:
                self.save_data(bci_data, indx, -1,self.
                  FREQ_STRUCT[str(Atrbt_aux)]-1,239-self.
                  TIME_STRUCT[str(indx)])
        if self.swap >= 235:
            self.learn()
            self.swap = -10
        return 0


    def getDsrAttrSimple(self):
        max_freq = -1
        best_attr = -1
        for attr in self.FREQ_DETECTION_ARRAY:
            if max_freq < self.FREQ_DETECTION_ARRAY[attr]:
                max_freq = self.FREQ_DETECTION_ARRAY[attr]
                best_attr = attr
        return best_attr


    def log_error(self, msg):
        log.error(self.time() + ":[STD]:␣" + msg)

    def log_info(self, msg):
        log.info(self.time() + ":[STD]:␣" + msg)

    def time(self):
        return dtm.datetime.now().time().isoformat()
```

A.5 Storage

```
__author__ = 'Guillermo␣Sarasa␣Duran'
import pymysql as sql


import logging as log
import datetime as dtm
log.basicConfig(filename='example.log', level=log.DEBUG)
import pymysql


class DB:
    userid = -1
```

```
    log = 0
    host = 'localhost'
    port = 3306
    user = 'root'
    db = -1
    cursor = -1
    passwd = ''


    def __init__(self,subjct,session):
        self.subjct=subjct
        self.session=session

    def connect(self):
        self.db = pymysql.connect(host=self.host, port=self.
          port, user=self.user, passwd='root', db='EEGData')
        self.cursor = self.db.cursor()
        return self.cursor

    def close(self):
        return self.db.close()

    def insert(self, query):
        #print query
        self.connect()
        try:
            # Execute the SQL command
            self.cursor.execute(query)
            # Commit your changes in the database
            self.db.commit()
        except:
            # Rollback in case there is any error
            self.db.rollback()
            self.log_error("Error, wrong query")
        self.close()

    def select(self, query):
        self.connect()
        try:
            # Execute the SQL command
            self.cursor.execute(query)
            results = self.cursor.fetchall()
            self.close()
            return results
        except:
            # Rollback in case there is any error
```

```
            self.close()
            self.log_error("Error,␣wrong␣query")
            return -1

    def login(self, userid):
        self.log_info("Logging␣with␣id␣=␣" + userid + "...")

        self.userid = userid
        return self.log

    def logout(self):
        if self.log != 1:
            self.log_error("Error,␣can't␣logout")
            return -1
        self.log = self.strg.logout()
        return self.log

    def selectSession(self, user_id, session_id):
        query = "SELECT␣*␣FROM␣" + self.signal_table + "␣WHERE␣session_id='" + session_id\
                + "'␣AND␣user_id='" + user_id + "';"
        return self.select(query)


    def insertSignalData(self, data, user_id, session_id):
        parsed_data = user_id + "," + session_id + ","
        for i in range(0, data.__len__()):
            parsed_data = parsed_data + "," + data[i]
        query = "INSERT␣(" + parsed_data + ")␣INTO␣" + self.signal_table + "␣WHERE␣user_id␣=␣'" + user_id
        + "'␣AND␣session_id␣=␣'" + session_id + "';"
        return self.insert(parsed_data)


    def log_error(self, msg):
        log.error(self.time() + ":[BCI->StrgDriver->DB]:␣" + msg)

    def log_info(self, msg):
        log.info(self.time() + ":[BCI->StrgDriver->DB]:␣" + msg)

    def time(self):
        return dtm.datetime.now().time().isoformat()
```

A.6 Predict stream script

This is a simple script to test the train functionality.

```

import BCI.OpenBCI_Interface.open_bci_v3 as opbci
import BCI.Offline as ofl
import Application.OfflineAp  as ofp
import kernel as ks
import Signal_Treatment.Signal_Treatment_Driver as std
import Storage.DB as sdb


db = sdb.DB(2,5)
board = ofl.OfflineBCI("/home/guillers/TFG/TFG/Files/",
  (60,48,56,51,47,34))
board.start()
sign = std.signal(1,1,1,1,7,17,end=160)
DMY = ofp.offlineAp("/home/guillers/TFG/TFG/Files/")
Krnl = ks.BCI_kernel(board, sign, db , DMY)
Krnl.train_predict_stream()
```

A.7 Simple offline stream script

```

import BCI.OpenBCI_Interface.open_bci_v3 as opbci
import BCI.Offline as ofl
import Application.dummy  as dm
import kernel as ks
import Storage.DB as sdb


db = sdb.DB(2,1)
board = ofl.OfflineBCI("/home/guillers/TFG/TFG/Files/",
  (60,48,56,51,47,34))
board.start()
DMY = dm.dummy()
Krnl = ks.BCI_kernel(board, None, db, DMY)
Krnl.online_stream_signal()
```

A.8 GUI

```

import matplotlib.gridspec as gridspec
```

```

import pylab
from pylab import *


class gui_sampler:

    #self.update_c = 4.166667
    update_c = 4.1666666666666667

    def __init__(self, freq_update=4.1666666666666667, values=[0 for i in range(100)]):
        update_c = update_c
        self.values = values

    def grid_plotter(self,chtr_matrix, ini, end, avarage = None):
        i = 0
        array = [i for i in range(ini,end)]
        gs = gridspec.GridSpec(6,6)
        h = []
        a = 1
        for i in range(0,6):
            for j in range(0,6):
                if a == 1:
                    a = 0
                    h = subplot(gs[i,j])
                else:
                    h = subplot(gs[i,j], sharey=h)
                plt.plot(array, chtr_matrix[i][j])
                if avarage != None:
                    plt.plot(array, avarage)
                    plt.plot(array, chtr_matrix[i][j]-avarage)
        plt.show()

    #https://hardsoftlucid.wordpress.com/various-stuff/realtime-plotting/
    def real_time_plotter(self,getEEGData):
        xAchse=pylab.arange(0,100,1)
        yAchse=pylab.array([0]*100)

        fig = pylab.figure(1)
        ax = fig.add_subplot(111)
        ax.grid(True)
        ax.set_title("Online EEG signal")
```

```
        ax.set_xlabel("Time")
        ax.set_ylabel("Amplitude")
        ax.axis([0,100,-6,6])
        line1=ax.plot(xAchse,yAchse,'-')

        manager = pylab.get_current_fig_manager()

        self.values = []
        self.values = [0 for x in range(100)]



        def RealtimePloter(arg):

          CurrentXAxis=pylab.arange(len(self.values)-100,len(
            self.values),1)
          line1[0].set_data(CurrentXAxis,pylab.array(self.
            values[-100:]))
          ax.axis([CurrentXAxis.min(),CurrentXAxis.max()
            ,-200000,200000])
          manager.canvas.draw()


        def handle_data(arg):
          data = getEEGData()
          if data == []:
            print "ERROR"
            return
          print data[0]
          self.values.append(data[0])

        timer = fig.canvas.new_timer(interval=self.update_c)
        timer.add_callback(RealtimePloter, ())
        timer2 = fig.canvas.new_timer(interval=self.update_c)
        timer2.add_callback(handle_data, ())
        timer.start()
        timer2.start()

        pylab.show()
```

Appendix B

Simulation examples

The following picture is the result of an alternative simulation the C and $gamma$ values:

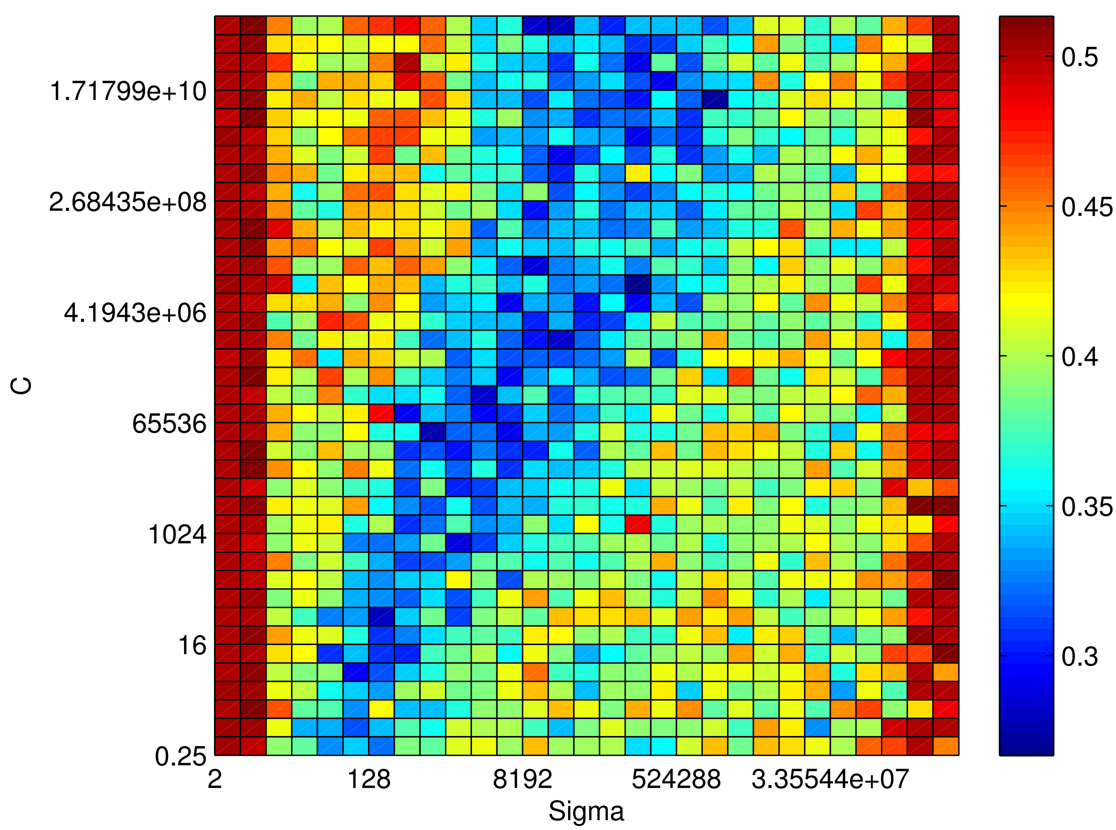

Figure B.1: Alternative simulation

The following picture shows the variance in the average signal in an alternative representation system:

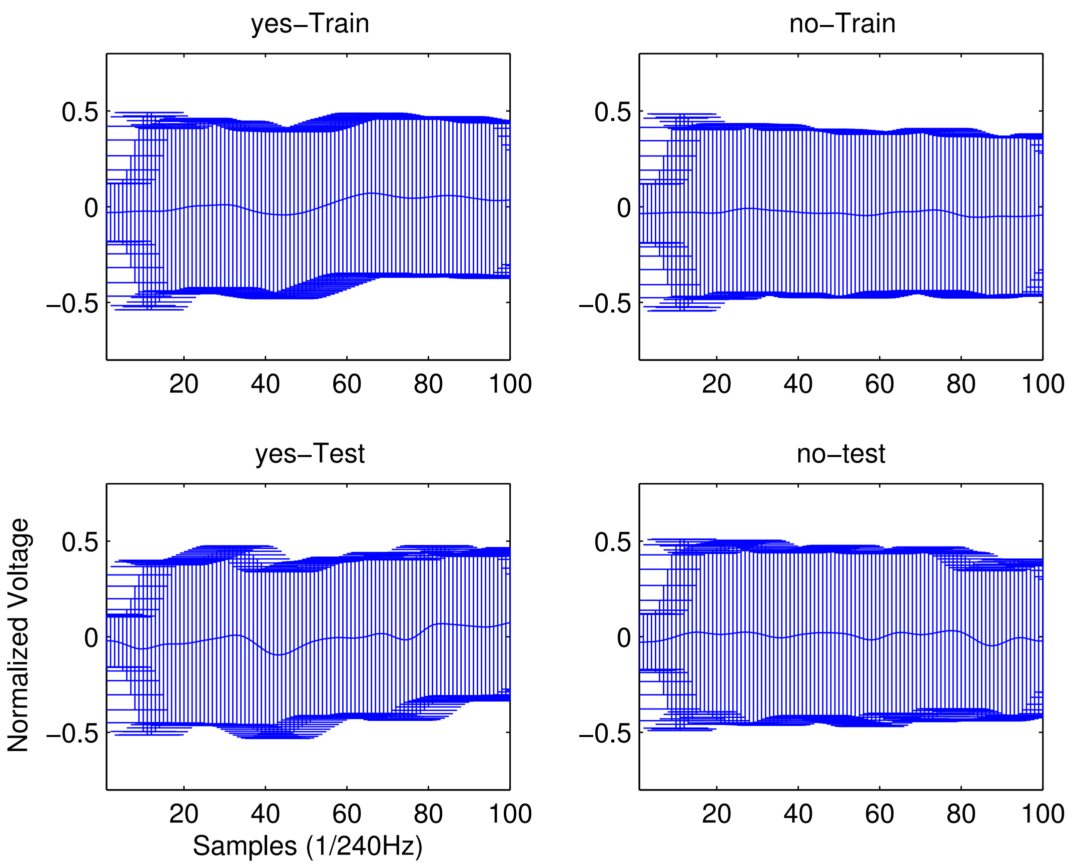

Figure B.2: Variance of the signal

Bibliography

[1] D. Luis Fernando Nicolás Alonso and Dr. D. Roberto Hornero Sánchez and Dr. D. Jaime Gómez Gil, *Clasificación de características de electroencefalogramas en sistemas Brain Computer Interface basados en ritmos sensoriomotores*, trabajo de fin de master, UNIVERSIDAD DE VALLADOLID.

[2] Claudia Nureibis Henríiquez Muñoz and Francisco de Borja Rodríguez Ortiz, *Estudio de Técnicas de análisis y clasificación de señales EEG en el contexto de Sistemas BCI (Brain Computer Interface)*, trabajo de fin de master, UNIVERSIDAD Autónoma de Madrid.

[3] Emanuel Donchin,Kevin M. Spencer and Ranjith Wijesinghe, *The Mental Prosthesis: Assessing the Speed of a P300-Based Brain–Computer Interface*, IEEE TRANSACTIONS ON REHABILITATION ENGINEERING, VOL. 8, NO. 2, JUNE 2000

[4] Simon Thorpe, Denis Fize and Catherine Marlot, *Speed of processing in the human visual system*, Centre de Recherche Cerveau & Cognition, UMR 5549, 31062 Toulouse, France

[5] Squires NK, Squires KC and Hillyard SA., *Two varieties of long-latency positive waves evoked by unpredictable auditory stimuli in man*, Electroencephalogr Clin Neurophysiol. 1975 April **38(4)**:387-401.

[6] Christoph Gugera, Shahab Dabana, Eric Sellers, Clemens Holznera, Gunther Krausza, Roberta Carabalonac, Furio Gramaticac and Guenter Edlingera *How many people are able to control a P300-based brain–computer interface (BCI)* Neuroscience Letters **462** (2009) 94–98

[7] Hiran Ekanayake, *P300 and Emotiv EPOC: Does Emotiv EPOC capture real EEG*December 25, 2010

[8] Peter Meinicke, Matthias Kaper, Florian Hoppe, Manfred Heuman and Helge Ritter *Improving Transfer Rates in Brain Computer Interfacing: A Case Study*, University of Bielefeld, Bielefeld, Germany

[9] C. Brunner, G. Andreoni, L. Bianchi, B. Blankertz, C. Breitwieser, S. Kanoh, C. A. Kothe, A. Lecuyer, S. Makeig, J. Mellinger, P. Perego, Y. Renard, G. Schalk, I. P. Susila, B. Venthur and G. R. Muller-Putz, *BCI Software Platforms*,

[10] Arnau Espinosa, Guenter Edlinger and Christoph Guger, *P300 Brain-Computer Interface Performance*, COGNITIVE 2013 : The Fifth International Conference on Advanced Cognitive Technologies and Applications

[11] Luis Fernando Nicolas-Alonso and Jaime Gomez-Gil, *Brain Computer Interfaces, a Review*, Department of Signal Theory, Communications and Telematics Engineering, University of Valladolid, Valladolid 47011, Spain

[12] *http://www.bbci.de/competition/iii*

[13] Magdalena Michalska, *OpenBCI: Framework for Brain-Computer Interfaces*,University of Warsaw Faculty of Mathematics, Informatics and Mechanics, September 2009

[14] Alain Rakotomamonjy and V. Guigue *BCI Competition III: Data Set II&mdash,Ensemble of SVMs for BCI p300 Speller*, IEEE Trans. Biomedical Eng., vol. 55, no. 3, pp. 1147-1154, Mar. 2008.

[15] Lagerlund TD1, Sharbrough FW, Jack CR Jr, Erickson BJ, Strelow DC, Cicora KM and Busacker NE., *Determination of 10-20 system electrode locations using magnetic resonance image scanning with markers.*, Electroencephalogr Clin Neurophysiol. 1993 Jan;86(1):7-14.

[16] J. Fernandez-Vargas, H.U. Pfaff, F.B. Rodriguez and P. Varona., *Assisted closed-loop optimization of SSVEP-BCI efficiency*, Frontiers in Neural Circuits 7:27. 2013

[17] Zachary Cashero, *COMPARISON OF EEG PREPROCESSING METHODS TO IMPROVE THE CLASSIFICATION OF P300 TRIALS*, Colorado State University, Fort Collins, Colorado Summer 2011

[18] Neng Xu, Xiaorong Gao, Bo Hong, Xiaobo Miao, Shangkai Gao and Fusheng Yang, *CI Competition 2003—Data Set IIb: Enhancing P300 Wave Detection Using ICA-Based Subspace Projections for BCI Applications*, EEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, VOL. 51, NO. 6, JUNE 2004

[19] Yijun Wang , Shangkai Gao and Xiaorong Gao, *Common Spatial Pattern Method for Channel Selection in Motor Imagery Based Brain-computer Interface* Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 5392 - 5395